ARC-PATH APPROACHES TO FIXED CHARGE NETWORK PROBLEMS

A THESIS

Presented to

The Faculty of the Division of Graduate Studies

by

Ui Chong Choe

In Partial Fulfillment

of the Requirement for the Degree

Doctor of Philosophy

in the School of Industrial and Systems Engineering

Georgia Institute of Technology

October, 1977

ARC-PATH APPROACHES TO FIXED

CHARGE NETWORK PROBLEMS

Approved:

Ronald L. Rardin, Chairman

John J. Jarvis

Atif S. Debs

Leon F. McGinnis

Vernon E. Unger

Date Approved by Chairman

10/21/77

ACKNOWLEDGMENTS

I gratefully acknowledgment the assistance, counsel, and friendship of my advisor, Dr. R. L. Rardin. Without his help, valuable suggestions, and deep insight, this research would not have been possible, and I would like to express my deep appreciation to him.

Special thanks are due to Dr. J. J. Jarvis. Dr. Jarvis has helped the author throughout the study by his valuable suggestions as a teacher and member of dissertation committee.

The other members of my dissertation committee, Drs. Debs, McGinnis and Unger, also provided encouragement and useful advice.

In addition to these faculty members, inestimable aid has also been provided by numerous persons. Special credits are due my brother-in-law, Mr. Byung D. Lee, Rev. Yoo, Byung C., Mr. Won K. Sun, Mrs. Chin S. Monroe and Mrs. Blanca Rardin for encouraging me through and giving me every possible assistance.

Finally, I wish to thank my wife Ha Ran, and my two children, Yu Suk and Yu Jin for their patience and understanding over the past three years. They have sacrificed much while I have been engrossed in this effort.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

SUMMARY

In this dissertation special problem structures related to the arc-path formulations of fixed charge network problems (FCNP) are investigated in the context of branch-and-bound procedures. A number of different problem formulations are derived from extensions of arc-path approach and the various formulations are compared. Identifiable structures in the formulations are exploited to yield a number of algorithmic procedures for solving the acyclic fixed charge network problems. The procedures are then integrated into a branch-and-bound procedure for FCNP drawing heavily on arc-path linear relaxations for bounds.

The proposed techniques for FCNP's are implemented in a computerized algorithm and tested on a set of randomly-generated waste-water test problems. Computational results support the viability of employing the augmented arc-path form for providing bounds in a branch-and-bound procedure.

CHAPTER I

INTRODUCTION AND LITERATURE REVIEW

The linear fixed charge network problem (FCNP) can be formulated as follows:

$$\begin{aligned} \min \quad & c^t x + f^t y \\ \text{s.t.} \quad & Ex = 0 \\ & Uy \geq x \geq \ell \\ & 1 \geq y \geq 0 \\ & y \text{ integer} \end{aligned}$$

where c and f are variable and fixed cost vectors respectively, x is an arc flow vector, y is a vector of logical variables which take on the value 1 when the corresponding x variable is positive and zero otherwise, ℓ is a vector of lower bounds on flows, U is a diagonal matrix of upper bounds u_j on arc flows, and E is the node-arc incidence matrix of a directed network. Numerous familiar problems in operations research, including the fixed charge transportation problem and the warehouse location problem take this form.

The above formulation can be called the node-arc formulation because it centers on the node-arc incidence matrix. An equivalent solution is obtained if the problem is formulated in the corresponding arc-path form as follows:

$$\min \quad d^t w + f^t y$$

$$\text{s.t.} \quad Uy \geq Pw \geq \ell$$

$$w \geq 0$$

$$1 \geq y \geq 0$$

$$y \text{ integer}$$

where w is a vector of flows along paths through the network, d is a vector of total variable costs along paths, P is the arc-path incidence matrix of the network, and y, ℓ and f are as above. Furthermore, the optimal solution to the arc-path formulation is unchanged when the redundant path capacity constraints

$$t_k y \geq P\, w_k \quad \text{for all paths } k$$

are added, where $t_k = \min_{\substack{\text{arcs } j \\ \text{in path } k}} \{u_j\}$ and P_k is the k^{th} column of P.

Most previous attention in operations research has been directed to the node-arc formulation of (FCNP) because of the simple and elegant structure of the matrix E. For example, special problem structures related to the group theoretic formulations of (FCNP) were investigated in the context of penalty-oriented branch-and-bound procedures by Rardin [39].

The potential value of other formulations centers on their ability to produce better bounds for a branch-and-bound algorithm for FCNP. To establish this branch-and-bound context for the ensuing chapters, a generalized branch-and-bound algorithm is briefly illustrated in Figure 1 and outlined here. To facilitate the discussion, define the function

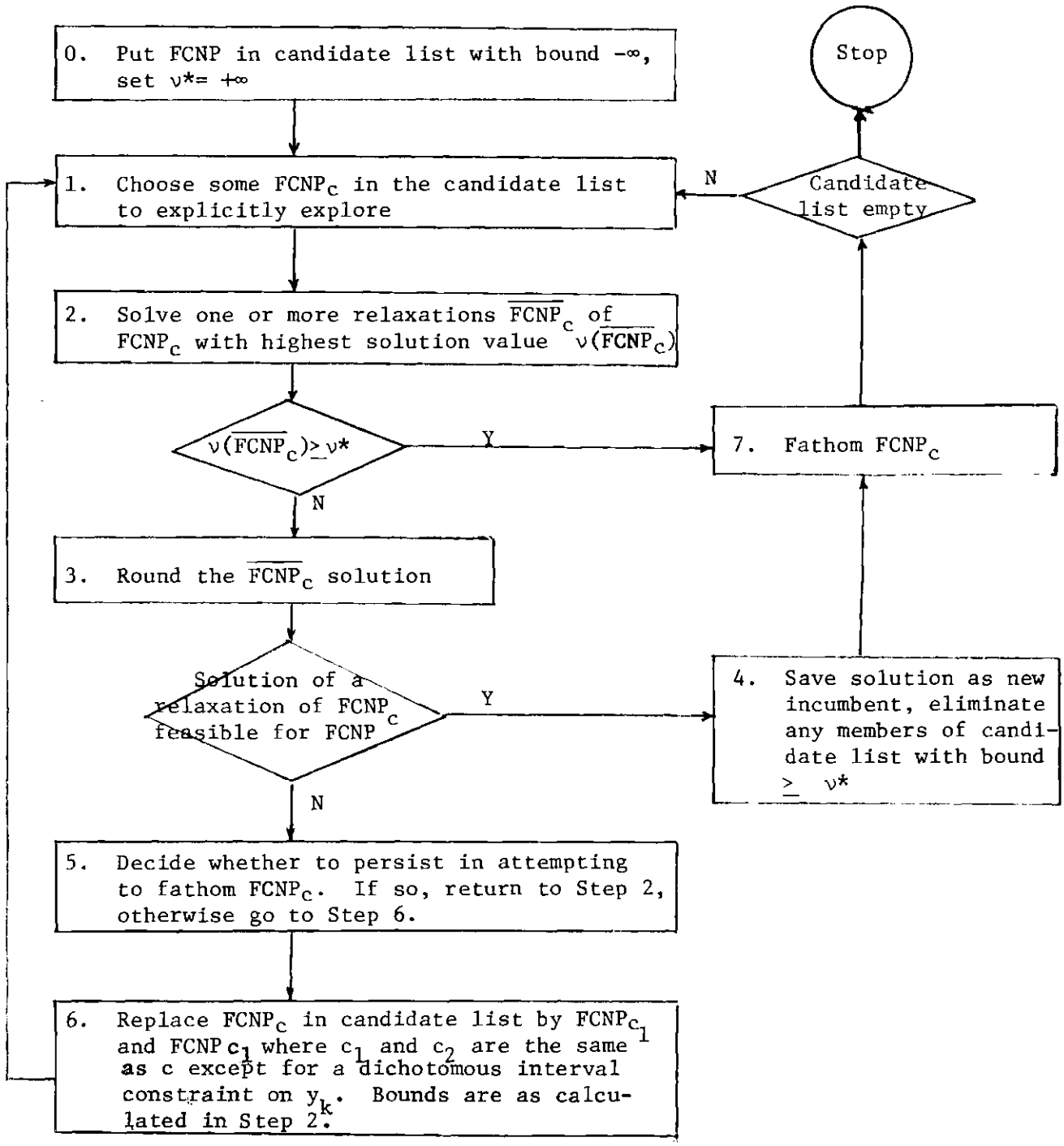


Figure 1. Flow Chart of General Branch-and-Bound Approach.

$\nu(.)$ to be the value of an optimal solution to the problem given as its argument ($= +\infty$ if none exists and $-\infty$ if the solution is unbounded).

The set of feasible solutions to (FCNP) can be partitioned into independent subsets by an enumeration which places additional constraints on integer variables. The unenumerated portion of (FCNP) can be represented by a list of candidate problems, ($FCNP_c$), each of which is (FCNP) with certain additional constraints. Each additional constraint stipulates that the value of one of the integer variables (y_j) must lie in a certain closed interval or take on fixed value. The best currently known feasible solution to (FCNP) is called the incumbent solution with incumbent solution value ν^*. In Step 1, some candidate problem ($FCNP_c$) which could still produce an optimal solution to (FCNP), is chosen to be explicitly explored. One or more relaxations are then solved in Step 2. If it is determined that ($FCNP_c$) could not yield a feasible solution to (FCNP) which is better than the incumbent solution, then ($FCNP_c$) is fathomed, i.e., eliminated from further consideration in Step 7. If a solution to a relaxation of ($FCNP_c$) is found to be feasible for (FCNP) then a new incumbent is saved at Step 4, and ($FCNP_c$) is fathomed; no solution to ($FCNP_c$) can produce a lower cost.

When a candidate problem is examined and not fathomed, the candidate problem is separated into two simpler candidate problems. In Step 6, a dichotomous interval constraint on the branching variable is added to produce the two new candidate problems. Returning to Step 1, the procedure is repeated until no candidate problems remain.

The successful application of a relaxation in a branch-and-bound

scheme can be seen to depend on the quality of the bounds at Step 2 and the ease of computing them. Larger bounds will reduce the number of candidate problems which must be explored, but computing efficiency is important since one must repeat the procedures over and over with different candidate sets.

Many integer and mixed integer programs have more than one formulation, i.e., more than one way to write the objective function and the constraint set (see Williams [47]). The different formulations will give the same solution value when solved as integer programs but may give different results when a relaxed version of the formulation is solved to obtain a bound.

The warehouse location problem provides one good example. The capacitated warehouse location problem is often stated as:

$$
\begin{aligned}
(\mathrm{WLP})_{\mathrm{NA}} \qquad \min \quad & \sum_{ij} c_{ij}x_{ij} + \sum_{i} f_i y_i & & \\
\text{s.t.} \quad & \sum_{j} x_{ij} \le S_i y_i & & i=1, \ldots, m \\
& \sum_{i} x_{ij} = D_j & & j=1, \ldots, n \\
& x_{ij} \ge 0 & & i=1, \ldots, m;\ j=1, \ldots, n \\
& y_i = 0 \text{ or } 1 & & i=1, \ldots, m
\end{aligned}
$$

where $\sum_{i} S_i > \sum_{j} D_j$

S_i and D_j are interpreted as the amount of supply at a potential warehouse or source, and the demand at a city or sink. The problem is called uncapacitated if the S_i are arbitrarily large numbers. Fixed charges are incurred in constructing (or opening) sources.

The above statement of the problem could be called the node-arc formulation because it centers on arc flows x_{ij}. An equivalent formulation, which could be called the arc-path formulation, may be stated:

$$
\begin{aligned}
& \min && \sum_i \sum_j c_{ij} x_{ij} + \sum_i f_i y_i && \\
(\mathrm{WLP})_{\mathrm{AP}} \quad & \text{s.t.} && \sum_j x_{ij} \le S_i y_i && i=1, \ldots, m \\
& && \sum_i x_{ij} = D_j && j=1, \ldots, n \\
& && x_{ij} \ge 0 && i=1, \ldots, m;\ j=1, \ldots, n \\
& && y_i = 0 \text{ or } 1 && i=1, \ldots, m \\
& && x_{ij} \le y_i \min \{D_j, S_i\} && i=1, \ldots, m;\ j=1, \ldots, n
\end{aligned}
$$

The difference here is the last set of constraints. Those constraints associate each x_{ij} with the corresponding y_i as opposed to dealing with the sum $\sum_j x_{ij}$. When the y_i are restricted to be integer, the added constraints are redundant, but the bound obtained from the relaxed problem with y_j merely in $[0,1]$ may be much different from that of $(\mathrm{WLP})_{\mathrm{NA}}$. One empirical study on eight warehouse location problems by Ramsay [38] showed that the optimal value of the linear relaxation in the arc-path case was always within 1% of the integer optimum. The corresponding relaxation of the node-arc form attained only 64-84% of the integer value. This improvement in bounds obtained from solving the linear relaxation of $(\mathrm{WLP})_{\mathrm{AP}}$ rather than that of $(\mathrm{WLP})_{\mathrm{NA}}$ has been also pointed out by Davis and Ray [8], Geoffrion and Graves [19], Geoffrion and McBride [21], and others. The potential for obtaining such improvement in more general FCNP's is the theme of this dissertation.

Literature on Exact Solution of Fixed Charge Network Problems

A great deal of research has been devoted to exact solution of the fixed charge transportation and the warehouse location problem cases, but much less attention had been given to general fixed charge network problems. A fixed charge transportation problem can be stated:

$$\min \quad \sum_i \sum_j (c_{ij}x_{ij} + f_{ij}y_{ij})$$

$$\text{s.t.} \quad \sum_j x_{ij} \leq S_i \qquad i=1,\ldots,m$$

$$\sum_i x_{ij} = D_j \qquad j=1,\ldots,n$$

$$x_{ij} \geq 0 \qquad i=1,\ldots,m;\ j=1,\ldots,n$$

$$x_{ij} \leq y_{ij} \min\{S_i, D_j\} \qquad i=1,\ldots,m;\ j=1,\ldots,n$$

where $\sum_i S_i = \sum_j D_j$

Here the S_i correspond to the maximum available supply at each supply node or source, and D_j is the amount of demand at each demand point or sink.

The important exact solution methods for fixed charge transportation problem were developed by Spielberg [43], who focused on a direct implementation of Bender's partitioning techniques [4], Murty [35] and Frank [17], who based their method on extreme point ranking methods, Gray [23], who employed branch-and-bound in combination with Hillier's [29] bound-and-scan method, and Tompkins [46] and Kennington [32], who used group theoretic methods. Complete literature surveys are provided in [17,23,32,46].

Among the most important exact solution methods for the warehouse

location problem are these reported by Efroymson and Ray [12], Davis and Ray [8], Spielberg [43], Ellwein [13], Bulfin and Unger [5,6], Hansen [24], Khumawala [33], Erlenkotter [14,15], Geoffrion and Graves [19], and Geoffrion and McBride [21]. Most of these exact procedures are essentially Land-Doig [34] branch-and-bound algorithms, with bounds drawn from linear programming relaxation of node-arc form. Efroymson and Ray were among the first to develop such approaches. Spielberg, Ellwein, and Bulfin and Unger combined them with Bender's partitioning [4]. Khumawala [33] devised an efficient updating procedure with respect to storage requirements and computing time, based on Efroymson and Ray [12]. Hansen [24] reported two implicit enumeration algorithms exploiting the concept of penalties.

The early work of Davis and Ray [8], and more recent developments by Erlenkotter, Geoffrion and McBride, and Geoffrion and Graves, employed methods which can be characterized as dual-based. Davis and Ray, and Geoffrion and Graves used dual solution approach to the arc-path form of (WLP) presented earlier. Strong bounds obtained from that formulation provided efficient enumerative procedures. Erlenkotter proceeds directly from the complementary slackness relationships for that strong relaxation and its dual. He devises a dual adjustment procedure which attempts to close the gap between the dual and the integer primal solutions by reducing the complementary slackness violations.

Geoffrion and McBride work with the lagrangian dual of $(WLP)_{NA}$ which can be stated as follows:

$$\max_{v_j} \left\{ \begin{matrix} \min \\ x_{ij} \geq 0 \\ y_i = 0 \text{ or } 1 \end{matrix} \left\{ \sum_i \sum_j c_{ij}x_{ij} + \sum_i f_i y_i + \sum_j v_j\left(\sum_i x_{ij} - D_j\right) \right\} \right\}$$

A general theory of lagrangian relaxation which has provided a unifying framework for several bounding procedures in discrete optimization, has been developed by Geoffrion [18] with particular emphasis on LP-based branch-and-bound. Geoffrion and McBride [21] demonstrated that the path capacity constraints are the faces of the convex hull of the lag-

$$x_{ij} \leq y_i \min\{D_j, S_i\}$$

rangian relaxation. That relaxation tends to be an extremely tight approximation of the overall problem.

The only known algorithmic developments for general fixed charge network problems are the work of Rardin [39], and Rardin and Unger [40]. Jarvis, Rardin, Unger, Moore and Schimpeler [31] applied these schemes to a regional waste-water system network problem formulated as a fixed charge network flow problem. The method employed is a penalty oriented branch-and-bound technique. Penalties are derived from the group theoretic point of view, but shown to be closely related to those of Driebreek [11], Beale and Small [3], and Tomlin [45]. All bounds and penalties are obtained from the optimal solution to the linear relaxation of FCNP in node-arc form.

Literature of Arc-Path Network Algorithms

A number of theoretical and applied investigations have appeared in the literature on arc-path approaches in continuous (nonfixed charge) network problems. Since the number of variables (paths) in this formulation is usually too large to be dealt with explicitly, a specialized computing scheme for generating columns of incoming nonbasic variables is required. Ford and Fulkerson [16] suggested a simplex computation for an arc-path formulation of the maximal multicommodity network flow problem. In 1966, Tomlin [44] showed that the minimum cost multicommodity flow problem could be formulated in both node-arc and arc-path forms, leading to very large equivalent linear programs. Jarvis [30] formulated the maximal multicommodity flow problem in both node-arc and arc-path form and showed that when Dantzig-Wolfe decomposition [7] is applied to the arc-path form, the resulting master and subproblems become precisely those described by Ford and Fulkerson [16]. The methods used by both Tomlin and Jarvis can be viewed as revised simplex procedures with a special column generation scheme. Wollmer [48] demonstrated an extension of the case of above where more general constraints were considered in the generalized multicommodity flow problem. Efficiency of the procedures derives from the fact that the column generation method involves only the solution of a shortest path problem. By careful allocation of all dual multipliers and costs to arcs it can be shown that the length of a path is exactly its adjusted cost in the arc-path linear program.

Purpose of the Dissertation

As briefly reviewed in the previous sections, the status of research relevant to arc-path approaches to fixed charge network problem can be summarized as follows:

- Arc-path forms have been successful in warehouse location problems, but have not been extended to general fixed charge network problems.
- Some theory employing the efficient procedures of large scale linear programming has been developed for solving some arc-path formulations through column generation, but such techniques have not been considered in the context of fixed charge problems.

The purpose of this dissertation is to link these two areas of research by developing branch-and bound procedures for FCNP's which uses the linear relaxation of arc-path solution (with path capacity constraints) as their principal bound.

CHAPTER II

PROPERTIES OF FIXED CHARGE NETWORK FORMULATIONS

Usnig the notation of Chapter I, the linear fixed charge network problem (FCNP) can be formulated as either the node-arc formulation or the arc-path formulation, respectively

$$(\text{FCNP})_{\text{NA}} \qquad \begin{aligned} \min \quad & c^t x + f^t y \\ \text{s.t.} \quad & Ex = 0 \\ & Uy \geq x \geq \ell \\ & 1 \geq y \geq 0 \\ & y \text{ integer} \end{aligned}$$

and

$$(\text{FCNP})_{\text{AP}} \qquad \begin{aligned} \min \quad & d^t w + f^t y \\ \text{s.t.} \quad & Uy \geq Pw \geq \ell \\ & w \geq 0 \\ & 1 \geq y \geq 0 \\ & y \text{ integer} \end{aligned}$$

It has already been noted that the optimal solution to the arc-path formulation is unchanged when the redundant path capacity constraints

$$t_k y \geq P_k w_k \quad \text{for all paths } k$$

are added. In the discussion to follow $(\text{FCNP})_{\text{AP}}$ with these path capacity constraints added will be referred to as the augmented arc path formulation or $(\text{FCNP})_{\text{AAP}}$.

Most standard linear-programming based branch-and-bound algorithms for FCNP have relied on the familiar node-arc formulation. The purpose of this chapter is to develop properties of $(FCNP)_{AP}$ and $(FCNP)_{AAP}$ which will facilitate their use in such a branch-and-bound scheme.

Relations Between the Continuous Node-arc and Arc-path Formulations

In terms of the notation of Chapter 1, the corresponding node-arc and arc-path formulations of the ordinary (continuous) network flow problem are

(NA)
$$\begin{aligned} \min \quad & c^t x \\ \text{s.t.} \quad & Ex = 0 \\ & Ul \geq x \geq \ell \\ & x \text{ unrestricted} \end{aligned}$$

and

(AP)
$$\begin{aligned} \min \quad & d^t w \\ \text{s.t.} \quad & Ul \geq Pw \geq \ell \\ & w \geq 0 \end{aligned}$$

Deo [9] and Seshu and Reed [42] summarize basic relations between the above forms in terms of node-arc incidence matrix and arc-circuit incidence matrix (of which P is a part), both for undirected and directed graphs. Two important results connecting bases of the two problems are provided below.

Theorem 2.1

A is a basis submatrix of the node-arc incidence matrix of a directed graph if and only if the columns of A correspond to the branches of a tree.

Proof: See Seshu and Reed [42].

Theorem 2.2

B is a basis submatrix of the arc-circuit matrix of a directed graph if and only if the columns of B correspond to the set of chords for some tree of the graph.

Proof: See Seshu and Reed [42].

Stated briefly the above theorems imply:

- Basis submatrices of node-arc incidence matrices are in one-to-one correspondence with the trees of the graph.

- Basis submatrices of arc-circuit incidence matrices are in one-to-one correspondence with complements of trees of the graph.

A corresponding result for dual solutions to the two forms is as follows:

Theorem 2.3

An optimal bound constraint dual solution (α,β) in the (NA) formulation provides an optimal dual solution of (AP) formulation.

Proof: Take duals of both problems as follows:

			dual variables
	min	$c^t x$	
	s.t.	$Ex = 0$	π
(NA)		$-x \geq -U1$	α
		$x \geq \ell$	β
		x unrestricted	

Note that since $\ell \geq 0$, we can assume $x \geq 0$.

$$\max \quad -1^t U^t \alpha + \ell^t \beta$$

			dual variables
(Dual of NA)	s.t.	$E^t_\pi - \alpha + \beta \le c$	
		$\alpha,\beta \ge 0$	
	min	$d^t w$	
(AP)	s.t.	$-Pw \ge -Ul$	α
		$Pw \ge \ell$	β
		$w \ge 0$	
	max	$-1^t U^t \alpha + \ell^t \beta$	
(Dual of AP)	s.t.	$-P^t \alpha + P^t \beta \le d$	
		$\alpha,\beta \ge 0$	

Multiply (Dual of NA) by P^t on the left,

$$\begin{aligned} \max \quad & -1^t U^t \alpha + \ell^t \beta \\ \text{s.t.} \quad & P^t E^t_\pi - P^t \alpha + P^t \beta \le P^t c \\ & \alpha,\beta \ge 0 \end{aligned}$$

Observe $P^t E^t = (EP)^t = 0$ (see for example [9,42])

and $P^t c = d$

Thus we have the following

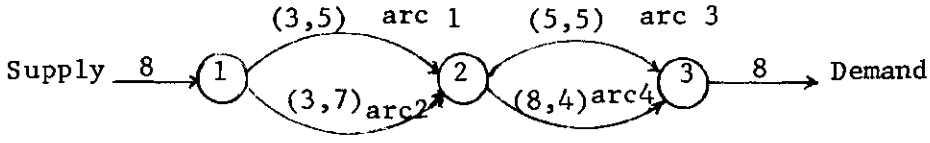
$$\begin{aligned} \max \quad & -1^t U^t \alpha + \ell^t \beta \\ \text{s.t.} \quad & -P^t \alpha + P^t \beta \le d \\ & \alpha,\beta \ge 0 \end{aligned}$$

Thus an (α,β) solving (Dual of NA) solves (Dual of AP). Q.E.D.

We have mentioned repeatedly that the above two formulations are equivalent. To show their equivalence constructively, consider first starting with an arc-path solution w. Summing the path flows along the arcs will produce the corresponding arc flow, i.e.

$$\Sigma P_j w_j = x_i \qquad \text{all paths along arc } i$$

The resulting x is clearly feasible for (NA). When starting with a node-arc solution, a set of corresponding path flows can be constructed by decomposing arc flows. Consider the problem



label = (fixed charge cost, upper bound).

Suppose it is desired to construct an arc-path solution given the following node-arc solution:

$x_1 = 1$, $x_2 = 7$, $x_3 = 5$ and $x_4 = 4$ with objective value = 14.6.

The procedure for decomposing x into a w is to pick a path and place flow on the path until some x value is exhausted. Then continue with another path until another x is exhausted, etc. One such solution is $w_1 = 0$, $w_2 = 1$, $w_3 = 5$ and $w_4 = 2$ where path 1 passes through arc 1 and 3, path 2 through arc 1 and 4, path 3 through arc 2 and 3 and path 4 through arc 2 and 4. Notice that while this solution is feasible for (AP) it is not basic. The sum of (+1) times the columns for w_1 and w_4 plus (-1) times the columns for w_2 and w_3 is a zero vector.

Bases of the Arc-Path Linear Relaxation

Consider the following linear relaxation of $(FCNP)_{AP}$,

$$\begin{aligned} \min \quad & d^t w + f^t y \\ \text{s.t.} \quad & Uy \geq Pw \geq \ell \\ & w \geq 0 \end{aligned}$$

$$1 \geq y \geq 0$$

It will be convenient to slightly rewrite the constraints, so that only one copy of P is included. Observe that only rows for fixed charge arcs are truly subject to [1]

$$P^j w \leq u_j y_j$$

and lower bounds on such arcs must be zero. Non-fixed charge arcs are constrained by either

$$P^j w = \ell_j$$

or

$$P^j w \leq u_j$$

If a nonfixed charge arc were subject to both upper and lower limits it could be replaced by an arc with lower and upper bounds equal to ℓ_j, plus a parallel arc with lower bound 0 and upper bound $u_j - \ell_j$. Thus it is sufficient to consider only nonfixed charge arcs with exactly one of the above constraint forms. It follows that the above linear relaxation may be assumed to have the following constraint tableau (after including slacks):

w	y	s_1	s_2	RHS	Remarks
P^f	$-U^f$	I		0	Fixed charge arcs
P^n			I	$U^n 1$	Nonfixed charge arcs (Inequality arc)
P^e				ℓ	Nonfixed charge arc (Inequality arc)

1. Here P^j is row j of P and u_j is the j^{th} diagonal element of U.

Here P^f, P^n and P^e are relevant parts of P and U^f, U^n are corresponding submatrices of U.

Denote the above form by $(\overline{FCNP})_{AP}$. Now consider what variables might span given rows in a basis of the above tableau. The following diagram enumerates the cases:

	Basic w	Basic Slacks y=0	Basic Slacks y=1	Basic Slacks non-fixed	Basic y	Nonbasic y y=0	Nonbasic y y=1	Nonbasic Slacks y=0	Nonbasic Slacks y=1	Nonbasic Slacks non-fixed	y basic	Non-basic w
Path	$P_1^{f=0}$					$-U_2, 0$		I				
variable	$P_1^{f=1}$						$-U_3, 0$		I			
w	$P_1^{n=u}$									I		Q_1
is												
basic	P_1^e											
Slack	$P_2^{f=0}$	I				$0, -U_4$						
s	$P_2^{f=1}$		I				$0, -U_5$					Q_2
is												
basic	$P^{n\neq u}$			I								
y is basic	P_3				$-U_1$						I	Q_3

Figure 2. Complete $(\overline{FCNP})_{AP}$ Tableau

Note that the submatrix of rows where w_k are basic, P_1, is composed of

- Rows where fixed charge arcs have both y nonbasic at 0 and a nonbasic slack

- Rows where fixed charge arcs are capacitated with y nonbasic at 1 and a nonbasic slack

- Rows of capacitated non-fixed charge arc

- Equality rows

The submatrix of rows where slacks are basic, P_2 is composed of

- Rows for fixed charge arcs with y_j nonbasic at 0

- Rows for fixed charge arcs with y_j nonbasic at 1

- Rows for uncapacitated nonfixed charge arcs.

Finally there is a group of rows, P_3, where the y_j variables are basic.

The following theorem and corollaries demonstrates that all possible cases are included in the above enumeration.

Theorem 2.4

Bases of $(\overline{FCNP})_{AP}$ may be rearranged into the following form,

Basic w	Basic Slack	Basic y
P_1		
P_2	I	
P_3		$-U_1$

Proof. The proof will proceed by construction from the bottom of an arbitrary basis matrix.

1. Collect at bottom of the matrix all rows corresponding to basic y_j. No slacks in the same rows may be basic because the column for y_j is a multiple of the column for s_j. Thus, the P_3 part of the basic has the above form.

2. Collect in the middle of the matrix all rows corresponding to basic slacks. By construction, no coefficients are present in the columns for basic y_j, and the P_2 portion of the basis has the specified form.

3. Collect all remaining rows at the top of the matrix. By construction these rows contain nonzero coefficients only for basic w_k, and the P_1 portion of the matrix is as above. Q.E.D.

Corollary 2.5

Given any basis of $(\overline{FCNP})_{AP}$, rows and columns of the problem can be rearranged to obtain the form of Figure 2.

Proof. The proof again proceeds from the bottom. Rows corresponding to P_3 are specified in Theorem 2.4.

A slack may be in the basis, and any corresponding y_j out, because either the y_j is nonbasic at value 0, or it is nonbasic at value 1, or the arc is a nonfixed charge one having no y_j. These are exactly the three cases depicted in the P_2 section of Figure 2.

When neither a slack for some row nor its y_j is in the basis, the cause may be any of the three cases enumerated for P_2 or the fact that the row is an equality one. The latter rows have nonzero coefficients only in w columns. It follows that the four cases for P_1 in Figure 2 are also exhaustive, and the tableau in the figure is complete. Q.E.D.

Corollary 2.6

After rearranging the tableau corresponding to any basis of $(FCNP)_{AP}$ into the format of Figure 2, the corresponding basis inverse and nonbasic portion of the updated simplex tableau are as shown in

Figure 3.

Proof. The corollary follows by direct inversion of the basis illustrated in Figure 2 and multiplication of the tableau in that figure by the resulting inverse. Q.E.D.

Any linear-programming based procedure for solving $(\overline{FCNP})_{AP}$ via bound from $(\overline{FCNP})_{AP}$ will require an efficient algorithm for obtaining an optimal solution to $(\overline{FCNP})_{AP}$. Moreover, any simplex-oriented algorithm for $(\overline{FCNP})_{AP}$ will revolve around bases of the initial problem tableau. The potential value of the above theorem and corollaries lies in the fact that bases of $(\overline{FCNP})_{AP}$ have an identifiable form which can be exploited algorithmically. Specifically, all elements of the basis inverse and updated simplex tableau for $(\overline{FCNP})_{AP}$ can be easily constructed from original problem data if the submatrix (P_1^{-1}) is known explicitly. Other entries in Figure 3 are either matrix products of P_1^{-1} and original problem data or terms involving the diagonal matrices U_k and U_k^{-1}. Thus it is possible to carry out simplex operations on $(\overline{FCNP})_{AP}$ while explicitly storing only the matrix P_1^{-1}. Since the number of rows of $(\overline{FCNP})_{AP}$ is as large as the number of arcs in the original problem, and the size of P_1 may be much smaller, this observation may result in substantial storage savings.

Partially Uncapacitated Networks and the Size of P_1

It is often the case that fixed charge network problems do not have true capacities on all fixed charge arcs. A fixed charge is assessed whenever there is a nonzero flow on such arcs, but there is no upper

BASIC INVERSE			UPDATED NONBASIC MATRIX						
Basic w	Basic Slack	Basic y	Nonbasic y				Nonbasic Slack		Nonbasic w
			y = 0		y = 1		Fixed Nonfixed	Fixed	
P_1^{-1}			$-P_1^{-1}\begin{pmatrix} U_2 \\ 0 \\ 0 \end{pmatrix}$	0	$-P_1^{-1}\begin{pmatrix} U_3 \\ 0 \\ 0 \end{pmatrix}$	0	$P_1^{-1}\begin{pmatrix} I \\ 0 \end{pmatrix}$	0	$P_1^{-1}Q_1$
$-P_2P_1^{-1}$	I		$P_2P_1^{-1}\begin{pmatrix} U_2 \\ 0 \\ 0 \end{pmatrix}$	$-U_4\begin{pmatrix} I \\ 0 \\ 0 \end{pmatrix}$	$P_2P_1^{-1}\begin{pmatrix} U_3 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -U_5 \\ 0 \end{pmatrix}$	$-P_2P_1^{-1}\begin{pmatrix} I \\ 0 \end{pmatrix}$	0	Q_2 $-P_2P_1^{-1}Q_1$
$U_1^{-1}P_3P_1^{-1}$		$-U_1^{-1}$	$-U_1^{-1}P_3P_1^{-1}\begin{pmatrix} U_2 \\ 0 \\ 0 \end{pmatrix}$	0	$-U_1^{-1}P_3P_1^{-1}\begin{pmatrix} U_3 \\ 0 \\ 0 \end{pmatrix}$	0	$U_1^{-1}P_3P_1^{-1}\begin{pmatrix} I \\ 0 \end{pmatrix}$	$-U_1^{-1}$	$-U_1^{-1}Q_3$ $+U_1^{-1}P_3P_1^{-1}Q_1$

Figure 3. Basis Inverse and Updated Simplex Tableau for $(\overline{FCNP})_{AP}$

limit on flow. In such cases it is customary to create an artificial upper limit u_j which is greater than or equal to the maximum possible flow through the arc. The following definition isolates arcs which are truly constrained.

Definition

An arc of (FCNP) is said to be critical if either it is equality constrained or its upper bound, u_j, is less than the maximum feasible flow through arc j.

Using the concept of critical arcs it will now be possible to establish a specific bound on the size of the P_1 matrix which was the center of the results in the previous section.

Theorem 2.7

The number of rows in the P_1 portion of the $(\overline{FCNP})_{AP}$ tableau in Figure 2 is less than or equal to the number of critical arcs in (FCNP).

Proof. By definition the number of critical arcs in $(\overline{FCNP})$ is the number of lower and upper capacity constraints which can be simultaneously binding in $(\overline{FCNP})_{AP}$. But those are exactly the rows of $(\overline{FCNP})_{AP}$ in which it is possible to have both y_j and s_j nonbasic, i.e., the P_1 rows in the Figure 2. Thus the number of critical rows is the maximal size of P_1. Q.E.D.

Corollary 2.8

The maximal number of paths in a basis of $(\overline{FCNP})_{AP}$ is equal to the number of critical arcs in (FCNP).

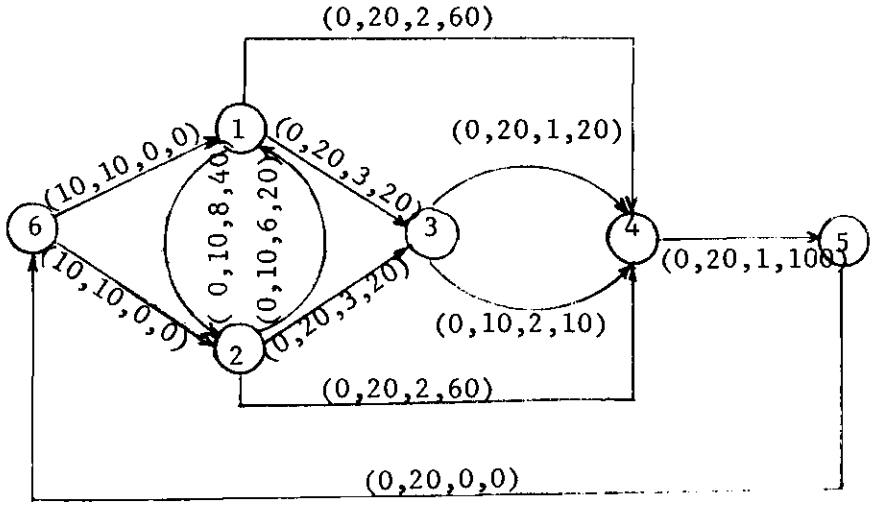
Proof. This corollary follows directly from Theorem 2.7 because P_1 is a square matrix. Q.E.D.

From the above results it is obvious that there is a very large potential savings of storage if $(\overline{FCNP})_{AP}$ is solved while keeping explicitly only P_1^{-1} . For example a capacitated warehouse location problem with m sources and n demand points has m + n critical arcs. However, if every source can ship to every demand point, it has m + n + mn arcs (rows in $(\overline{FCNP})_{AP}$). Thus the constructions of the previous section make it possible to solve an m + n + mn row linear program while explicitly keeping a matrix inverse of at most m + n by m + n.

The Augmented Arc Path Formulation

At the beginning of this chapter three formulations of FCNP were presented, the node-arc form, $(FCNP)_{NA}$, the arc-path form, $(FCNP)_{AP}$, and the augmented arc-path form $(FCNP)_{AAP}$. The construction treated earlier assure that the integer formulation satisfy $\nu(FCNP)_{NA} = \nu(FCNP)_{AP}$ and their linear relaxations also have $\nu(\overline{FCNP})_{NA} = \nu(\overline{FCNP})_{AP}$. Also, since the path capacity constraints which distinguish $(FCNP)_{AAP}$ are redundant in the integer formulation it must be true that $\nu(FCNP)_{AP} = \nu(FCNP)_{AAP}$. Moreover, clearly $\nu(\overline{FCNP})_{AAP} \geq \nu(\overline{FCNP})_{AP}$.

The example of Figure 4 shows that it is possible to have $\nu(\overline{FCNP})_{AP} < \nu(\overline{FCNP})_{AAP}$; that is, strict improvement is obtained when the path capacity constraints are included. For this case, the linear relaxation of the node-arc form yields an optimal value, $\nu(\overline{FCNP})_{NA} = \nu(\overline{FCNP})_{AP} = 220$. The augmented arc-path form (i.e. with path capacity constraints) has $\nu(\overline{FCNP})_{AAP} = 250$. Since the true optimum (i.e. $\nu(FCNP)$) is 260, it is clear that the augmented arc path formulation provides a much better bound than the node-arc one.



label = (lower bound, upper bound, variable cost, fixed charge).

Figure 4. Example Network with

$$\nu(\overline{FCNP})_{AP} < \nu(\overline{FCNP})_{AAP}$$

Theorem 2.9 summarizes the above observations about the three problem formulations.

Theorem 2.9

The following relations hold between the three formulations of FCNP presented at the beginning of this chapter.

$$\nu(\overline{FCNP})_{NA} = \nu(\overline{FCNP})_{AP} \leq \nu(\overline{FCNP})_{AAP}$$

$$\leq \nu(FCNP)_{AAP} = \nu(FCNP)_{AP} = \nu(FCNP)_{NA}$$

Moreover, the two inequalities may either or both be strict.

Because of the potential for bound improvement one would like to observe in the augmented arc path formulation a basis structure like the P_1,P_2 and P_3 scheme of the AP formulation. In AAP problems of even

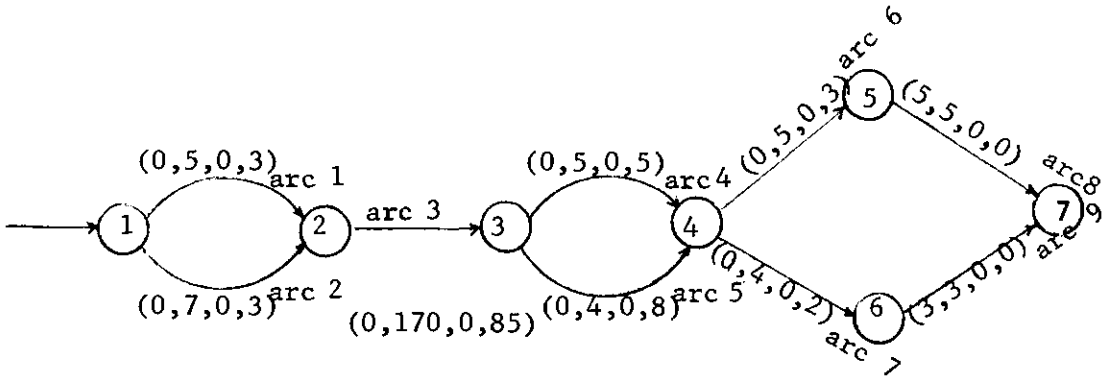
moderate size, there are thousands of path capacity constraints.

Various methods have been developed for implicitly representing constraints of the form $x_i \leq x_j$ in linear programs. Glover [22] and Schrage [41] term such constraints variable upper bounds (VUB); variable x_i is said to have a variable upper bound x_j. VUB constraints are common in a number of LP formulations, including fixed charge problems. In terms of basis representation, the procedure suggested by Schrage [41] corresponds to considering the slack of a VUB row as basic in its own row if it is basic at all. If a VUB slack is not basic, then the upper-bounded variable in the VUB row must be basic. Schrage says that x_j <u>is in the family</u> if the constraint $x_j \leq x_k$ is binding where one refers to x_j as the <u>child</u> variable and x_k as the <u>parent</u> variable.

Neither Schrage nor Glover treats the case when the same variable may be the child of more than one parent. Unfortunately, that is the case which arises in $(\overline{FCNP})_{AAP}$; the same w_k serves as the child of many y_j.

In order for the Schrage/Glover approach to extend to $(\overline{FCNP})_{AAP}$ it is necessary that no simultaneously binding pair of path capacity constraints, or combination of an arc row and a path capacity constraint involve the same w_k. In that case one member of the binding pair must be treated as a normal constraint, i.e., it cannot be dealt with implicitly in the Schrage/Glover manner.

Unfortunately, cases with such simultaneously binding constraints do occur. The Figure 5 gives an example,



label = (lower bound, upper bound, variable cost, fixed charge cost)

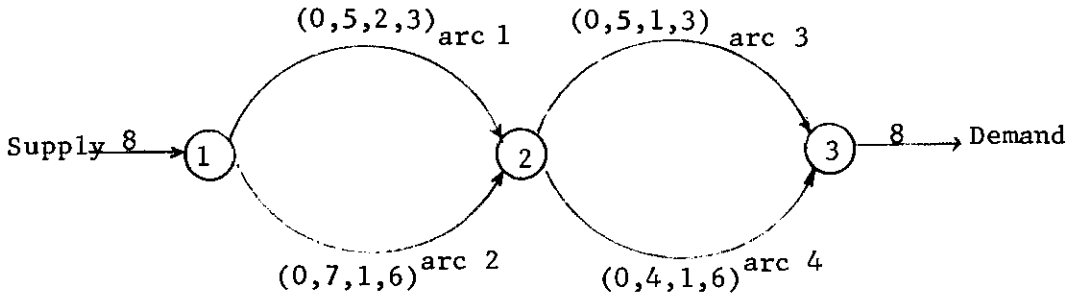
Figure 5. Network with Multiple Binding Path Capacity Constraints

In the unique solution to $(\overline{FCNP})_{AAP}$ for this problem all but one of the 8 path capacity constraints (with path variables $w_1, w_2, w_3, w_5, w_6, w_7$, and w_8 through arc 3) are simultaneously binding. Here path 1 passes through arc 1, 3, 4, 6, and 8, path 2 through arc 2, 3, 4, 7, and 9, path 3 through arc 2, 3, 5, 6, and 8, path 5 through arc 2, 3, 4, 6, and 8, path 6 through arc 1, 3, 4, 7, and 8, path 7 through 1, 3, 5, 6, and 8 and path 8 through arc 2, 3, 5, 7, and 9. The fact that the solution is truly affected by the presence of such constraints is illustrated by the decreasing value of the optimal solution as each is relaxed. The overall solution of 43.75 declines to 43.21 as the constraint with w_8 is deleted. Successive deletion of the constraints for w_7 and w_6 yields further reductions to 36.24 and 27.75.

Parallel Arc System

It is sometimes the case that (FCNP) includes parallel arc systems, i.e., sets of arcs connecting the same to nodes and oriented in the same

direction. Figure 6 is an example of such a network. Arcs 1 and 2, and arcs 3 and 4 form parallel systems.



label = (lower bound, upper bound, variable cost, fixed cost)

Figure 6. Parallel Arc System Example

The cost structure of these two systems are illustrated in Figure 7. When costs have no observable pattern, as is the case with the arc 3-4 system, no general statements can be made. However, parallel systems most often arise as piecewise linear approximations to concave functions. If the parallel arc cost structures do form a concave function, as in the 1-2 system, several properties can be derived. The next definition formally identifies this concave case.

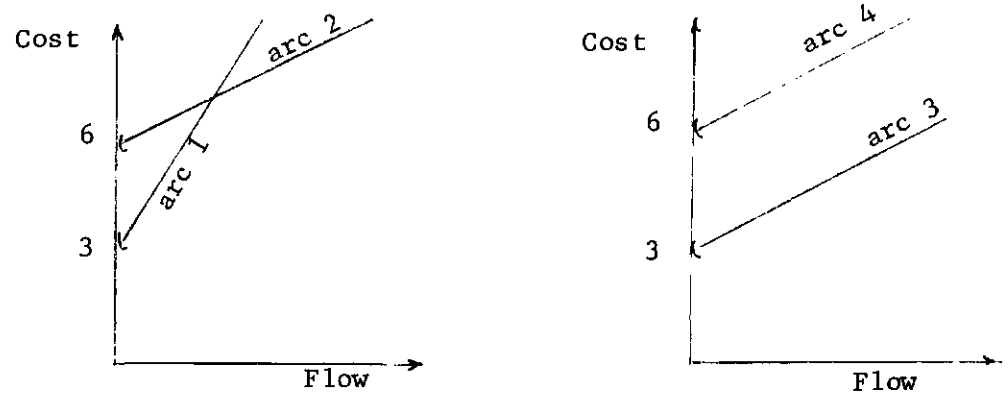


Figure 7. Cost Structures of Parallel Arc Examples

Definition

A parallel arc system in (FCNP), involving arcs in the index set J is said to be a concave parallel arc system if $u_J = \max_{j \in J} \{u_j\} \geq$ the maximum feasible flow through all arcs in J, and costs c_j and f_j, together with upper bounds u_j satisfy the following relationship:

$$f_k + c_k u_k \geq \min_{\substack{j \in J \\ j \neq k}} \{f_j + c_j u_k\}$$

for all k with $u_j < u_J$.

In simple terms the above definition merely requires that upper bounds not destroy concavity, i.e., that the cost of any parallel segment at its upper bound not be strictly smaller than the cost of all other segments at the same point.

The importance of concave parallel systems arises when considering the case where only one member of a parallel system may feasibly have positive flow, i.e. when (FCNP) is subject to additional generalized upper bound (GUB) constraints of the form

$$\sum_{j \in J} y_j \leq 1.$$

In (FCNP) such constraints are redundant for concave systems. If flow were placed on two segments in any solution, that solution could be improved while remaining feasible, by shifting all flow to the one arc with least c_j (see example Jarvis et al [31] for details). An optimal solution to $(\overline{FCNP})_{NA}$ is determined by minimizing the "effective cost" $(c_j + f_y/u_j)$ on the arcs (see for example Rardin and Unger [40] for a proof). For concave parallel arc systems, this implies all flow will

be placed on the one arc with minimum effective cost. Thus the GUB constraints are also redundant in the node-arc linear relaxation of (FCNP).

Since $\nu(\overline{FCNP})_{NA} = \nu(\overline{FCNP})_{AP}$, it follows that GUB constraints or concave parallel arc systems may also be ignored in the arc-path relaxation of (FCNP). However, the example of Figure 4 can be used to show that GUB constraints may affect the solution to the augmented arc-path relaxation $(\overline{FCNP})_{AAP}$. In that example $\nu(\overline{FCNP})_{NA} = \nu(\overline{FCNP})_{AP} = 220$ and $\nu(\overline{FCNP})_{AAP} = 250$. The two arcs connecting nodes 3 and 4 form a concave parallel system. When a GUB constraint is added on y_j for those arcs, $\nu(\overline{FCNP})_{AAP}$ increases to 260. The following theorem summarizes all the above results.

Theorem 2.10

Side constraints of the form $\sum_{j \in J} y_j \leq 1$ on concave parallel arc systems indexed by sets J may be disregarded in the problems (FCNP), $(\overline{FCNP})_{NA}$ and $(\overline{FCNP})_{AP}$. However, such constraints may be binding in the formulation $(\overline{FCNP})_{AAP}$.

CHAPTER III

SHORTEST ALLOWABLE PATHS

In Chapter I, it was noted that the methods used by Tomlin [44] and Jarvis [30] to solve arc-path formulations can be viewed as revised simplex procedures with a special column generation scheme; specifically nonbasic column generation involves the solution of a shortest path problem. The essential property on which this method depends is that all elements of the adjusted cost "$c_j - z_j$" on a path be allocable to arcs, i.e., the adjusted cost for any path should be the sum of preassigned weights on the arcs which make up the path.

Consider now that in $(\overline{\text{FCNP}})_{\text{AAP}}$ dual multipliers α_k^j have been assigned to each path capacity constraint

$$w_k \leq t_k y_j.$$

As before, dual multipliers on the main arc constraints will be denoted by π_j and the (variable) cost on arcs as c_j. If all α_k^j are 0, the adjusted cost of a nonbasic column <u>can</u> be calculated as the sum of arc "lengths" $c_j - \pi_j$. However, if any α_k^j are positive, the correspondence breaks down. The α_k^j are identified with particular paths and thus cannot be grouped into arc lengths.

Observe on the other hand that only a relatively few paths can have positive α_k^j. Whan a w_k is nonbasic (at value 0) the associated path capacity constraints are nonbinding. Thus the α_k^j can

become positive on w_k only when it becomes basic. In a column generating scheme for arc-path formulations, paths which becomes basic are explicitly known; the much more numerous nonbasic ones are not. It follows that positive values on α_k^j can be handled if an algorithm is devised which treats separately paths that are explicitly known and those that are unknown. The purpose of this chapter is to develop such a procedure for acyclic networks. In particular, an algorithm is developed which finds the shortest or allowable not forbidden path in the graph. Such paths will be the unknown ones in $(\overline{FCNP})_{AAP}$, with known paths forbidden and treated outside the shortest path scheme.

Rationale of an Algorithm

Consider an acyclic network with nodes numbered so that the presence of arc (i,j) in the network implies $j > i$. In Dijkstra's shortest path algorithm [10] for such problems, one maintains a π_i which represents the shortest distance from node 1 to node i. At each iteration the algorithm advances to the next-numbered node and labels all nodes it leads to according to

$$\pi_j = \min\{\pi_j, \pi_i + c_{ij}\}$$

For the $(\overline{FCNP})_{AAP}$ case a similar idea will be followed, but two categories of information must be retained. It will be convenient to think of them in terms of a flag. At the top of the flag is the eagle (the analog of the π in Dijkstra's algorithm) which shows the shortest distance from node 1 to other nodes along allowable paths. The bars of flag fall below the eagle and describe sets of forbidden paths, arranged in

increasing distance from the origin. Each set is a collection of forbidden paths which follow an identical route from the origin to the labeled node and which have a distance from the origin less than the eagle distance. These bars of forbidden paths must be recorded because they are preferable routes to the eagle distance and they might be feasibly used at nodes beyond the present one if some arc is interposed which makes the path allowable.

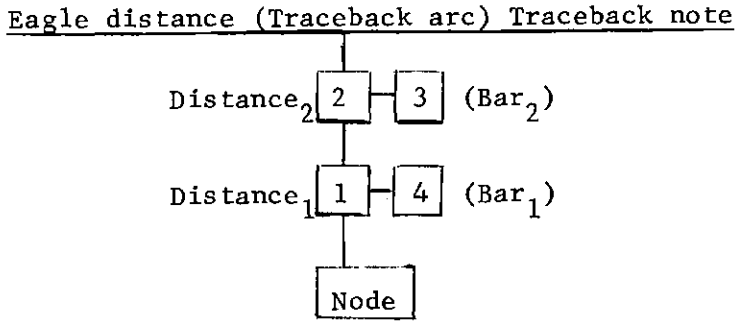
An algorithm using such information would move from node to node as with the Dijkstra procedure. Sets of forbidden paths which might yet lead to good allowable paths must be "passed across" arcs leaving the present node, and at the same time the eagle must be updated. The specific procedure to be presented starts with the lowest (shortest distance) bar of the present node's flag and scans the forbidden paths in each bar one by one. Those paths in a bar which must be passed on are divided into subsets that leave through the same arc. For this purpose temporary flags are established on each of the outgoing arcs. Bars in the temporary flags are subsets of forbidden paths in bars of the main flag which follow the same outgoing arc. Processing will terminate when all bars at a node have been divided into such subsets. Temporary flags will then be merged with the node flags at successor nodes.

To minimize processing it is important that forbidden paths which cannot improve on the eagle be deleted from flags as early as possible. This is accomplished primarily by retaining no bar in any flag that has bar distance greater than the eagle for that flag. However, some complexity arises in establishing eagle distances on temporary flags for

outgoing arcs. Specifically, the eagle on a temporary flag is the minimum of (i) any path which follows the eagle path to the present node and then the arc associated with the temporary flag and (ii) the distances associated with any bars of forbidden paths at the present node which include no path following the arc of the temporary flag. Such a rule can be implemented by introducing the concept of temporary flags having grown. A temporary flag grew on the previous bar scan if some part of the node bar was added as a bar in the temporary flag. If it did not grow, then the distance associated with the previous bar provides an eagle for the temporary flag; the partial path of the bar plus the arc of the temporary flag is an allowable path. Moreover, because node bars are scanned in ascending distance order, this allowable path is as short as any other one.

A System of Labels

As briefly mentioned above, at each node it will be useful to maintain a "flag" of information. The following illustrates a graphic label representation of this information:



Here the eagle distance of the flag is the analog of the π in Dijkstra's algorithm. Bars of the flag below the eagle are sets of forbidden paths,

arranged in increasing distance from the origin node. A forbidden path will never be a member of two such sets at one time. Thus bars can be defined in labels via a pointer at each node showing the first forbidden path number of the first bar, a pointer on each forbidden path showing the next path to the right or up, and a distance on forbidden paths showing the bar distance. In addition to the flags and distances already defined, two additional labels are associated (to the right) with each eagle. The first is in parenthesis and shows the inarc to this node by which the eagle distance was obtained. The second is either a zero (denoting that the eagle distance was derived from the eagle distance at the other end of the inarc) or a positive integer (specifying the number of the forbidden path which, in combination with the inarc forms the path that achieved the eagle distance). These last two labels will permit tracing the solution at optimality.

Labels for temporary flags on outgoing arcs would be identical to the above except that the inarc traceback label is unnecessary. Thus the total number of labels required to support the algorithm is 4 per node (eagle, traceback arc, traceback note, first bar path) plus 3 per arc (temporary flag eagle, temporary flag traceback note, temporary flag first bar path) plus 2 per forbidden path (next path, bar distance).

Formal Statement of the Algorithm

In terms of the above labels, a formal statement of a shortest allowable path algorithm can now be given. Steps of the algorithm are as follows:

Step 0. Initialize the eagle distance for each node at $+\infty$, establish a barless flag at nodes 2 through n. If these are no forbidden paths reset the eagle at 1 to 0. Otherwise, link all forbidden paths in a single bar flag at node 1. The bar distance for that bar is 0. Let the current node be node 1 and proceed to Step 1.

Step 1. Establish for each arc leaving the current node a temporary barless flag. The eagle distance for all such temporary flags is equal to the lowest bar distance of the flag at the current node (or the eagle if the current flag is barless). Also, establish the traceback label on each temporary tree as 0 or some path in the first bar of the current node's flag, according to the source of the temporary flag's eagle distance. Mark all temporary flags as having grown and proceed to Step 2.

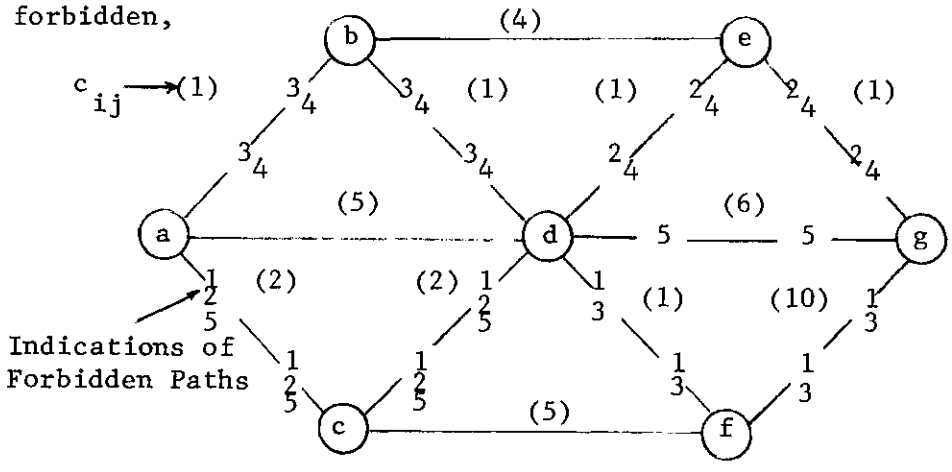
Step 2. If there are no remaining bars in the flag at the current node, go to Step 3. Otherwise, scan and process the next bar of forbidden paths in the flag of the current node and then go to Step 3. For each forbidden path in the bar, first determine which out arc it will use from the current node, i.e., the temporary flag to which it corresponds. If that temporary flag did not grow on the last execution of Step 2, skip to the next forbidden path. Otherwise, add the path to a bar in the temporary flag. If this is the first element of such a bar, also update eagle labels on the temporary flag. Specifically, set the eagle of the temporary flag equal to the next bar distance of the current node's flag (or to the eagle of the current flag if there is no next bar) and revise the traceback label accordingly.

Step 3. Add the length of the arc corresponding to all its distances and merge the temporary flag with the permanent flag at the successor end of the arc. In particular, choose as the new eagle distance of the merged flag the minimum of eagles of the flags being merged. Also, set traceback labels to correspond to the new eagle. Bars in the merged flag are all those in the flags being merged which have bar distance less than the new eagle distance. They are arranged in ascending order by bar distance. Go to Step 4.

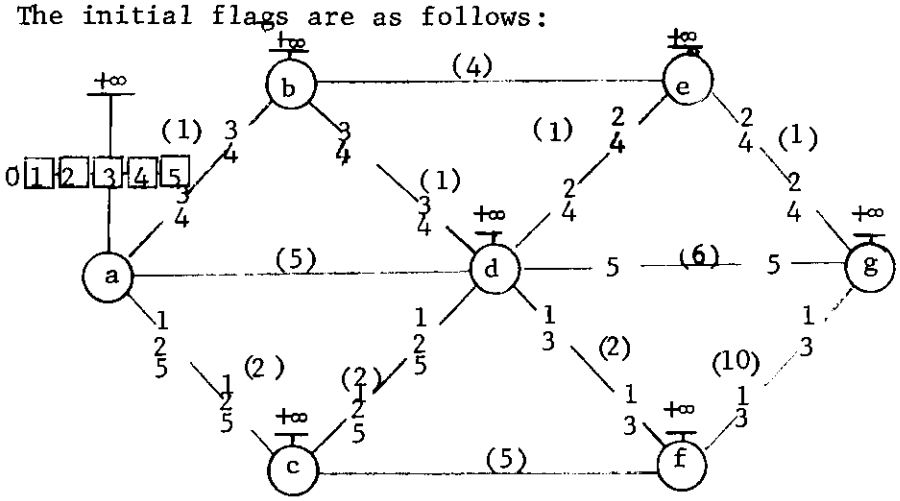
Step 4. Increase the current node number by 1. If it is still less than the number of nodes go to Step 1. Otherwise, trace out the optimal path via the traceback labels at each node. Specifically, begin with the last node and proceed along the inbound arc label associated with its eagle distance. If the traceback label at the last node is zero, repeat this process at the new node which is reached. Otherwise, backtrack along the forbidden path indicated by the traceback label until node 1 is reached.

A Numerical Example

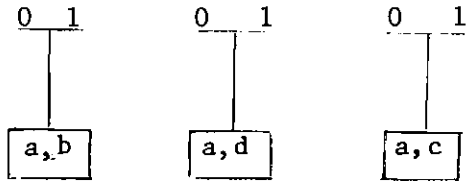
Consider the following problem where the 5 indicated paths are forbidden,



The initial flags are as follows:

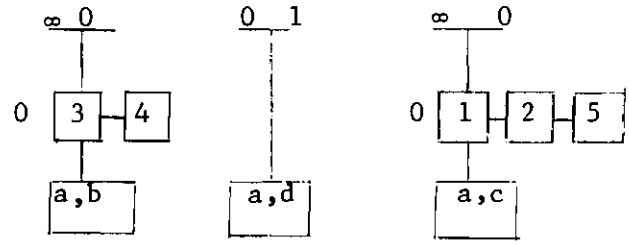


Processing of node (a) begins by creating three small flags corresponding to the three arcs leaving (a)

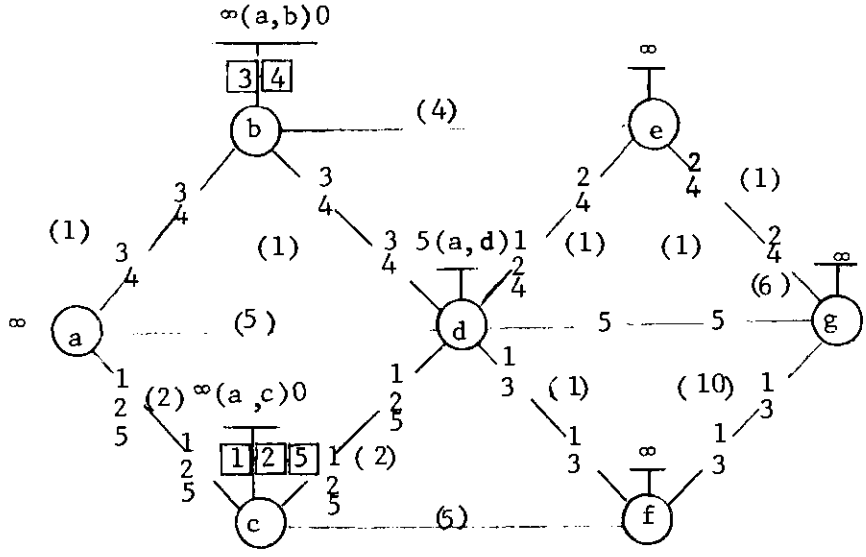


The eagle for each is the first bar distance at (a) and the code to the right of the eagle shows how to finish from node a.

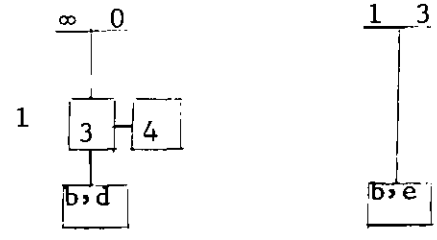
A scan of the one bar in the flag at (a) causes the three small flags to be revised as follows:



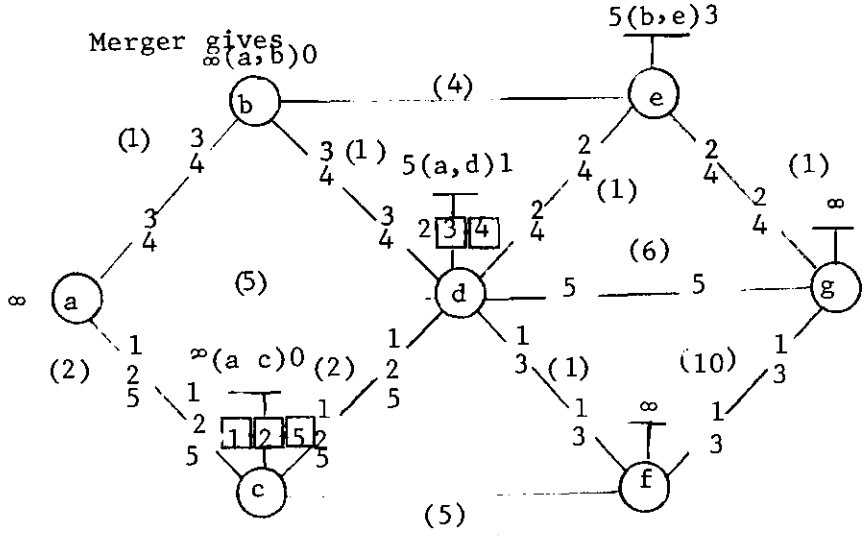
After merger with flags at successor nodes, these flags yield



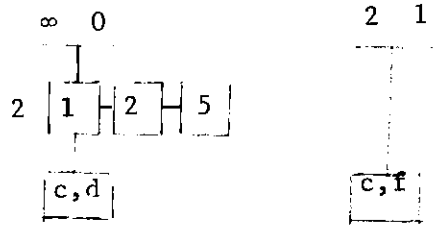
The flag at (b) subdivides

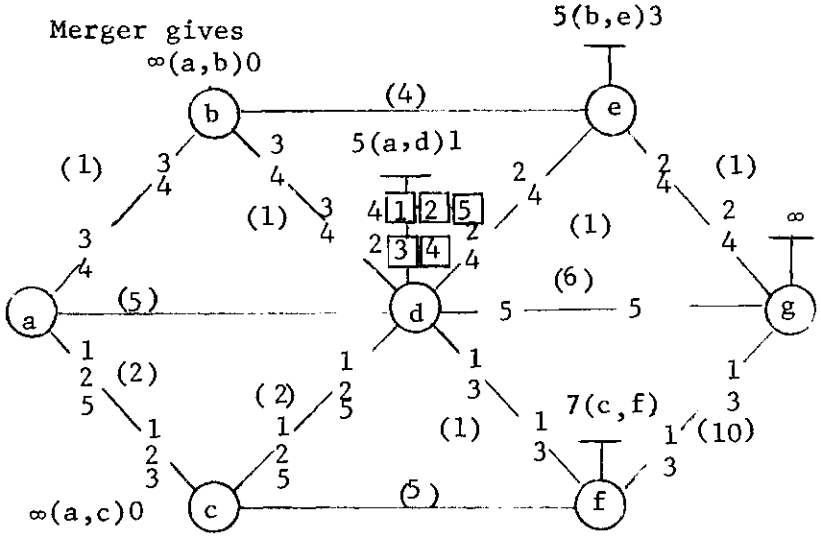


Merger gives

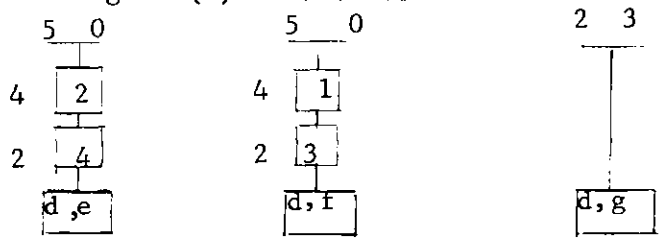


Then flag at (c) subdivides



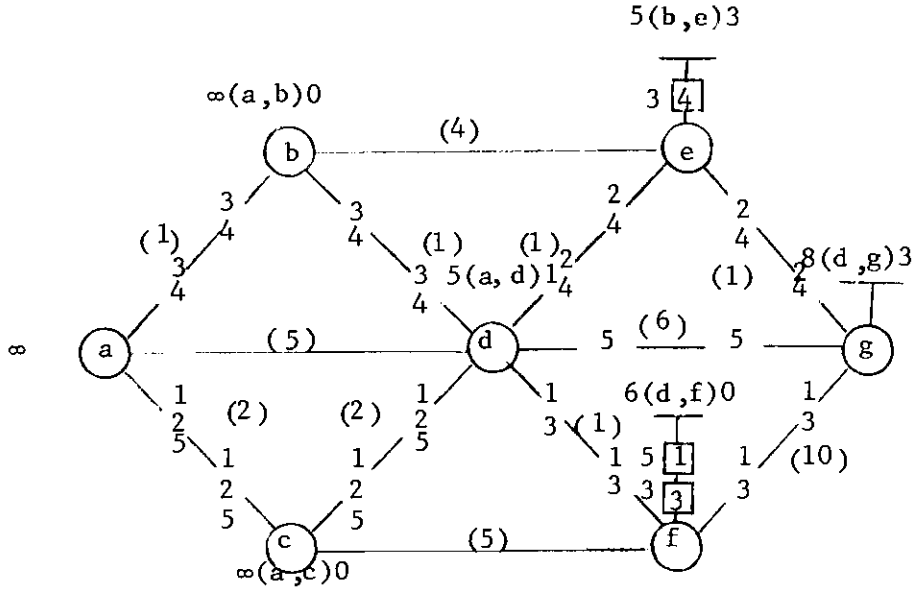


The flag at (d) subdivides



Notice that forbidden path 5 need not be carried in the flag for (d,g). When that flag failed to grow during the scan of the 3-4 bar, the combination of 3 or 4 and arc (d,g) was established as an allowable path. Since 5 falls in a higher bar it cannot improve on such an allowable path.

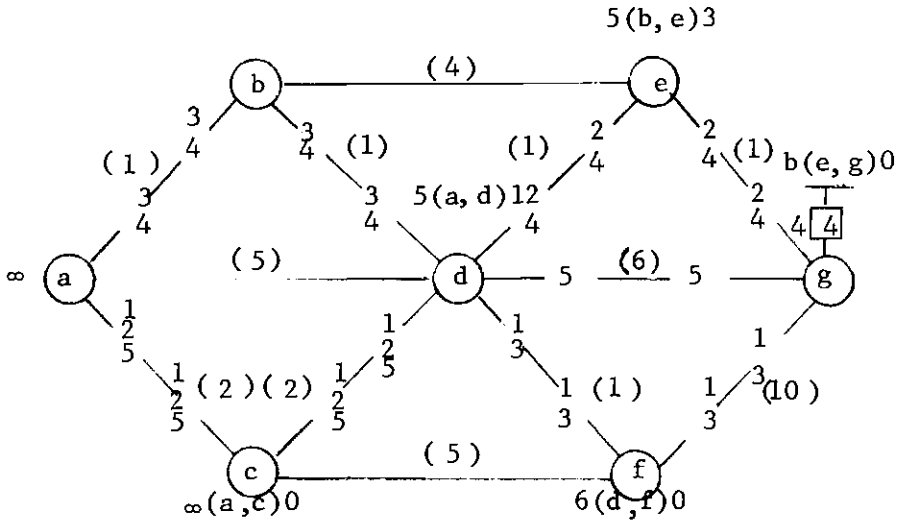
Merger gives



Notice that the upper bar of the flag for (d,e) was cut off in merger because its distance exceeded the minimum of the successor node eagle distance.

Both (e) and (f) have only one successor arc, so no breakdown is necessary.

Merger of these trees at (g) yields



The shortest path is forbidden path 4 with length 4. The shortest admissible path has length 6. It is identified by retracing the labels

saved with the eagles, first follow arc (e,g) and carry on with the eagle path, next follow arc (b,e) and complete along forbidden path 3. Thus the path follows forbidden path 3 to node b and then goes to e and g. The full path is a-b-e-g.

Proof of the Algorithm

It is clear that the above algorithm stops in a finite number of steps because there are only finitely many nodes in the network. Moreover, the traceback labels obviously do encode the path leading to the eagle distance at a node. Thus, to prove the solution produced by the algorithm is indeed the shortest allowable path it must only be verified that the final eagle distance at each node is the length of the shortest path from node 1 to the given node that does not fall entirely along a forbidden path.

The node 1 eagle distance of $+\infty$ when forbidden paths are present, and 0, when there are none, is clearly correct. Now assume the eagle distance is as required for nodes 1, 2, ..., k and consider any path p from node 1 to node k + 1 that is not entirely along forbidden path. It must be shown that the length of p is either greater than or equal to the eagle distance at node k + 1. Let node ℓ be the node touched by p immediately before node k + 1; there are two cases to be considered. First, the path from node 1 to node ℓ may not fall entirely along any forbidden path. In this case the length of p is greater than or equal to the eagle distance at ℓ plus the length of arc (ℓ, k + 1). Now when node ℓ was processed, the eagle distance of the temporary flag on (ℓ, k + 1) finished at most equal to the eagle at ℓ. Thus when the eagle

at k + 1 was set at the minimum of its previous value and the temporary flag eagle on (ℓ, k + 1) plus the length of (ℓ, k + 1), it was assured that it was less than or equal to the length of p.

The second case arises when p falls entirely along a forbidden path between node 1 and node ℓ. In this case the forbidden path p follows from 1 to ℓ was a member of some bar of the flag at ℓ when ℓ was processed. The length of p is the bar distance of that bar plus the length of arc (ℓ, k + 1). Now no forbidden path of that bar leads to node k + 1, or p would be entirely within a forbidden path. Thus in processing the flag at ℓ, the temporary flag for arc (ℓ, k + 1) failed to grow on the scan of that bar or some bar with lower bar distance. It follows that the eagle distance on the temporary flag for (ℓ, k + 1) finished with a value less than or equal to the distance from node 1 to node ℓ along p. After merger at node k + 1 it must be that the eagle at node k + 1 has length less than or equal to p. This complete the proof.

Some Computational Tests

In previous sections, a shortest allowable path algorithm was developed which is much like the Dijkstra algorithm for ordinary shortest path, requiring only one pass of the nodes and fixing the labels on one node at each iteration. When there are no forbidden paths the algorithm is exactly Dijkstra. As the number of forbidden paths grows, computation becomes more complex.

To evaluate empirically the efficiency of the procedure, a series of random shortest path problems were generated and solved on Georgia

Tech's CDC Cyber 76. Test problems had the characteristics shown in

Table 1. Characteristics of Shortest Path Problems

Arc Cost	Uniformly Distributed 0 to 100
Number of Arcs	500 or 1000
Number of Nodes	125 or 250
Number of Forbidden Paths	0 or 50 or 100
Random Number Seeds	An arbitrary real number

FORTRAN code for the test problem generator is included as Appendix A. The following provides a general outline of the process:

Step 1. Assign an arc to enter each node based on cell probabilities making all allowable (i,j) equally likely.

Step 2. Assign an arc to leave each node based on similar probabilities. (Note that Step 1 and Step 2 assure the network is connected).

Step 3. Assign the remaining arcs randomly until all required arcs are generated.

Step 4. Randomly generate lengths c_{ij} on each arc (i,j).

Step 5. Generate the required number of forbidden paths, making sure no duplicate paths are created.

A four factor experiment (number of arcs, number of nodes, number of forbidden paths and random number seeds) was conducted with this generator. Figure 8 shows the experimental layout. Note that levels of random number of seeds were nested under each level of arcs and nodes, but not under forbidden paths. Thus five independently-generated pro-

blems were solved at each level of arcs and nodes, and numbers of forbidden paths were varied using the same problems.

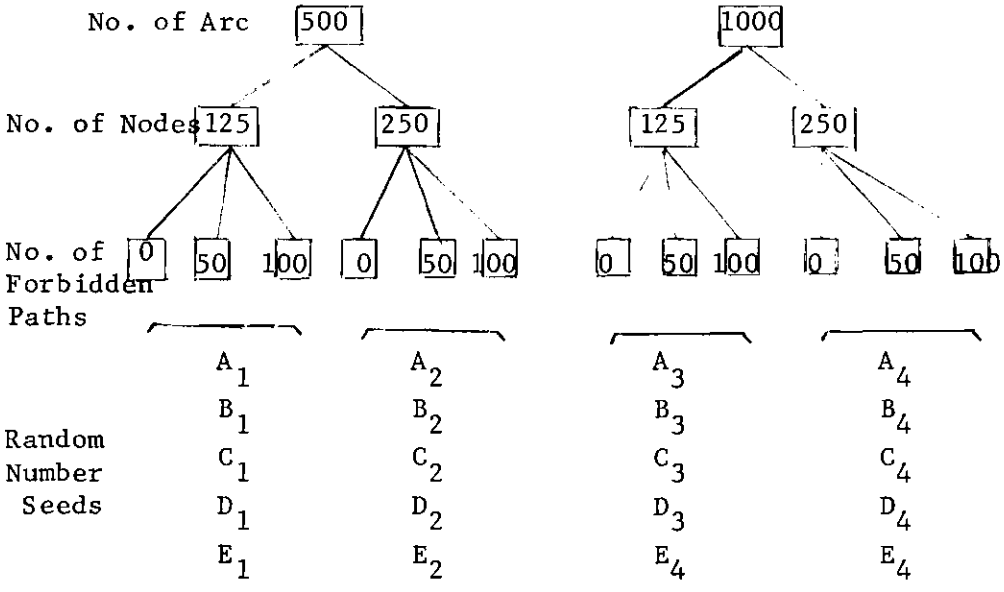


Figure 8. Experimental Layout for Shortest Path Tests

Table 2 presents the results of these experiments. Values shown in the table are the numbers of CPU seconds required to solve the problems on the Cyber 76. Table 3 summarizes the results by summing values for the five seeds used in each cell of the experiment. These summary results are also presented graphically in Figure 9.

As could be expected, the values in Figure 9 and Table 2 increase with increases in all main factors--numbers of arcs, numbers of nodes, and numbers of forbidden paths. However, all the times reported are relatively small.

Perhaps most important is the relation of the more complex shortest allowable path algorithm to the simpler shortest path method of

Table 2. Results of Shortest Path Experiments

FORBIDDEN PATHS	ARC (500)										ARCS (1000)									
	NODES (125)					NODES (250)					NODES (125)					NODES (250)				
	A_1	B_1	C_1	D_1	E_1	A_2	B_2	C_2	D_2	E_2	A_3	B_3	C_3	D_3	E_3	A_4	B_4	C_4	D_4	E_4
	Random No. Seeds																			
0	.069	.070	.072	.073	.068	.076	.060	.079	.077	.078	.119	.125	.098	.131	.130	.126	.140	.140	.136	.134
50	.094	.089	.101	.097	.098	.128	.123	.106	.119	.120	.165	.150	.161	.173	.136	.162	.169	.182	.170	.173
100	.128	.113	.114	.123	.128	.144	.158	.139	.162	.154	.204	.178	.195	.170	.184	.218	.230	.205	.208	.202

Table 3. Summary of CPU Time in Shortest Path Experiments

Arc	500		1000	
Node	125	250	125	250
0	.35	.37	.60	.68
Path 50	.48	.60	.79	.86
100	.61	.76	.93	1.06

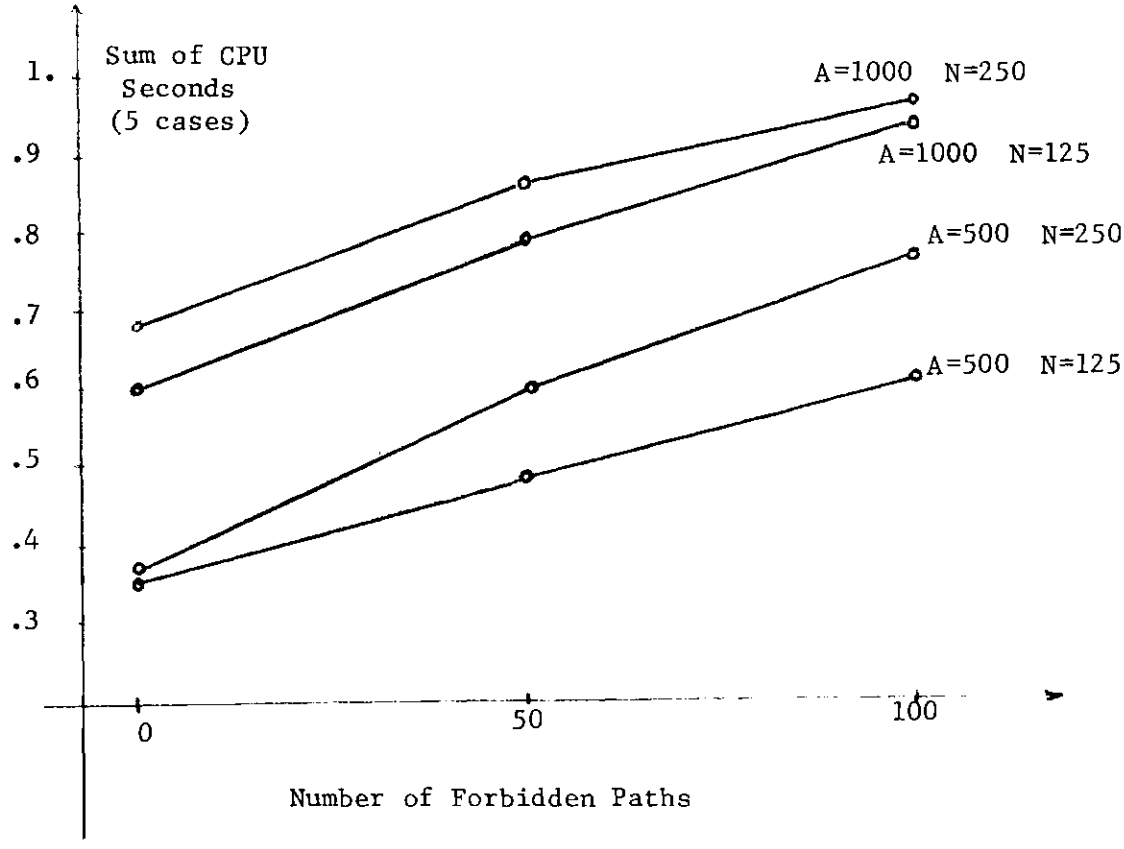


Figure 9. Total CPU Time for Five Problems by Numbers of Arcs, Nodes, and Forbidden Paths

Dijkstra. Values in Table 2 and Figure 9 for 0 forbidden paths are exactly the times to find shortest paths via Dijkstra's algorithm. It is clear that those values were consistently more than half ot the times to find shortest allowable paths when as many as 100 paths were forbidden. Since the Dijkstra algorithm has been successfully employed in a number of columns generating schemes discussed above, these results appear to suggest that the algorithm of this chapter is also satisfactory for such applications. Of course, the algorithm might prove unsatisfactory when large numbers of paths were forbidden. However, in the intended applications the forbidden paths will be those which are basic or otherwise explicitly retained in core. Thus their numbers are likely to be relatively small, and the proposed algorithm should be satisfactory.

CHAPTER IV

DEVELOPMENT OF AN ALGORITHM FOR SOLVING ACYCLIC FIXED CHARGE NETWORK PROBLEMS

In previous chapters a number of components of an arc-path approach to solution of fixed charge network problems (FCNP's) have been developed. Specifically, it has been demonstrated that

- The linear relaxation of FCNP in arc-path form with path capacity consttaints (i.e. $(\overline{FCNP})_{AAP}$) can yield better lower bounds for a branch and bound procedure than the other relaxation $(\overline{FCNP})_{NA}$ and $(\overline{FCNP})_{AP}$.
- Although bases of $(\overline{FCNP})_{AAP}$ can have a very complex structure, those of $(\overline{FCNP})_{AP}$ have a form amenable to a special purpose linear programming code that retains in core only a small submatrix of the overall basis inverse.
- For fixed values of the dual multipliers α_k^j, on path capacity constraints $w_k \leq t_k y_j$, a column generating scheme can be devised for acyclic (FCNP)'s which employs a shortest allowable path algorithm to implicitly cost out most nonbasic paths columns.

In this chapter these results are combined with known techniques to produce a complete branch-and-bound algorithm which solves (FCNP) using bounds derived from $(\overline{FCNP})_{AAP}$.

Decomposition Approaches

When one part of a linear program can be shown to have a special structure amenable to solution by specialized simplex procedures and other constraints add complications which destroy the special structure, it is common in mathematical programming to consider decomposition approaches. Such approaches solve the structured portion of the whole problem as a subproblem, and employ some interfacing procedure to produce a final solution which also satisfies constraints not included in the subproblem.

The previous analysis of this dissertation suggest such an approach could be appropriate for solving $(\overline{FCNP})_{AAP}$. The relaxation $(\overline{FCNP})_{AP}$ can be solved as a subproblem, with the path capacity constraints which distinguish $(FCNP)_{AAP}$ enforced through a decomposition technique.

The immediate problem which presents itself is the size of the "master problem" of path capacity constraints. Since there is one such constraint for each fixed charge arc along each path of the network, the number of path capacity constraints can become massive for even a small FCNP. Typical decomposition approaches--including Dantzig-Wolfe decomposition [7], Benders' partitioning [4], and Balas' infeasibility pricing [1]--require the explicit solution of the master problem between iterations of the subproblem. Thus to apply such methods it would be necessary to perform simplex operations on a linear program of the same order of size as the number of path capacity constraints. Unless very specialized procedures could be devised, it appears impractical to consider such a scheme.

A Subgradient Scheme for Obtaining Dual Multipliers

A closely related idea to formal decomposition is direct search for dual multipliers on the path capacity constraints. One such approach which has proved useful in recent integer programming research is based upon movement in the directions of subgradients. First proposed by Poljak [37], this method of optimization was explored by Held and Karp [26,27] as a method of obtaining a good upper bound for a branch-and-bound approach to the symmetric traveling-salesman problem, and by Bazaraa and Goode [2] for the asymmetric case. A paper by Held, Wolfe and Crowder [28] establishes the validity of the method for a wide class of mathematical programs and reports computational experience for assignment problems, symmetric traveling-salesman problems, and multi-commodity maximum flow problems. Neebe and Rao [36] report significant results for an algorithm for the solution of the m-medium problem--basically a network location problem closely related to the assignment problem. A paper by Hearn and Lowe [25] used the subgradient method to solve three general classes of minimax location problems.

Consider a mathematical programming problem in the form

$$\max \theta(\delta), \quad \delta \in S$$

where

$$\theta(\delta) = \min_k \{g_k + \delta^t \cdot v_k\}$$

and define the set $V(\delta)$ to be the set of v_k at the point (s) where the minimum is obtained, i.e.

$$V(\delta) = \{v_k : g_k + \delta^t \cdot v_k = \theta(\delta)\}$$

$V(\delta)$ will usually be a singleton; when it is, the function $\theta(\delta)$ is differentiable at δ, and the single member of $V(\delta)$ is the gradient $\nabla\theta(\delta)$. If $V(\delta)$ is not a singleton, then θ is not differentiable at δ, but the notion of differentiability can be extended; each v_k is a <u>subgradient at δ</u> and satisfies

$$\theta(\delta') \leq \theta(\delta) + v_k^t \cdot (\delta'-\delta) \quad \text{for every } \delta' \in S$$

since $\theta(\delta)$ is piecewise linear (when S is polyhedral) and concave.

Given an initial starting vector δ_o and a sequence of scalar step-sizes $\{\Delta_j\}$, a subgradient optimization proceeds by the interative scheme:

$$\delta_{j+1} = \delta_j + \Delta_j v_j \qquad \text{for } j=0, 1, \ldots,$$

where v_j is any subgradient at δ_j.

Such an approach will not generally yield a monotone sequence of $\theta(\delta_k)$. However, convergence to an optimal point (i.e. a point where the zero vector is a subgradient) can be shown (see Poljak [37]), provided two restrictions on $\{\Delta_j\}$ are satisfied:

$$\lim_{j\to\infty} \Delta_j = 0$$

$$\sum_{j=1}^{\infty} \Delta_j = \infty$$

Alternatively, using the choice

$$\Delta_j = \lambda_j \frac{\hat{\theta} - \theta(\delta_j)}{||v(\delta_j)||^2}$$

with $\hat{\theta} < \max \theta(\delta)$, $\varepsilon < \lambda_j \leq 2$ for some fixed $\varepsilon > 0$, the sequence $\theta(\delta_j)$ either converges to $\hat{\theta}$, or a point δ_j is obtained such that $\theta(\delta_j) \geq \hat{\theta}$.

Theoretically a subgradient procedure should stop when a δ is found such that the zero vector is a subgradient of θ at δ. However, it is necessary to somehow represent the entire subgradient set at each δ to determine whether 0 is a member. Since only one subgradient v_j is necessary to continue the search it is usually considered undesirable to actually apply the theoretical test. Instead, an ad hoc stopping rule is typically devised which stops when progress on maximizing θ has been too slow.

Application of such an approach to solution of $(\overline{FCNP})_{AAP}$ centers on the following equivalent statement of the problem in lagragian relaxation format:

$$\max_{\alpha_k \geq 0 \text{ for all } k} \left[\begin{array}{ll} \min & d^t w + f^t y + \sum_k \alpha_k^t (w_k P_k - t_k y) \\ \text{s.t.} & P^e w = \ell \\ & P^n w \leq U^n 1 \\ & P^f w \leq U^f y \\ & w \geq 0 \\ & 1 \geq y \geq 0 \end{array} \right]$$

An optimal solution (w,y) to the minimization part of the lagrangian relaxation establishes a constant $(d^t w + f^t y)$, which plays the role of g_k in the above discussion, and a subgradient vector

$$\begin{bmatrix} w_1p_1 - t_1y \\ w_2p_2 - t_2y \\ \vdots \\ w_kp_k - t_ky \\ \vdots \end{bmatrix}$$

which takes the role of v_k. The α_k are the lagrange multipliers analogous to δ above. For fixed α_k, the minimization is in the form of $(\overline{FCNP})_{AP}$, and the special structure of that form can be exploited. Appropriate steps along subgradient directions will eventually yield α_k which solve $(\overline{FCNP})_{AAP}$.

A Formal Subgradient-Oriented Procedure for $(\overline{FCNP})_{AAP}$

A formal algorithm implementing the above ideas is illustrated in Figure 10. Steps of the procedure are as follows:

Step 0. Determine if the lagrangian search should stop because the present subgradient is the 0 vector or because the ad hoc stopping rule has been satisfied. If so, return to branch-and-bound processing. Otherwise go to Step 1.

Step 1. Calculate the squared infeasibilities $||w_kp_k - t_ky||^2$ for path capacity constraints and use the result to compute the step size λ of the subgradient search. Go to Step 2.

Step 2. Revise the α_k on all w_k in the present explicit list according to

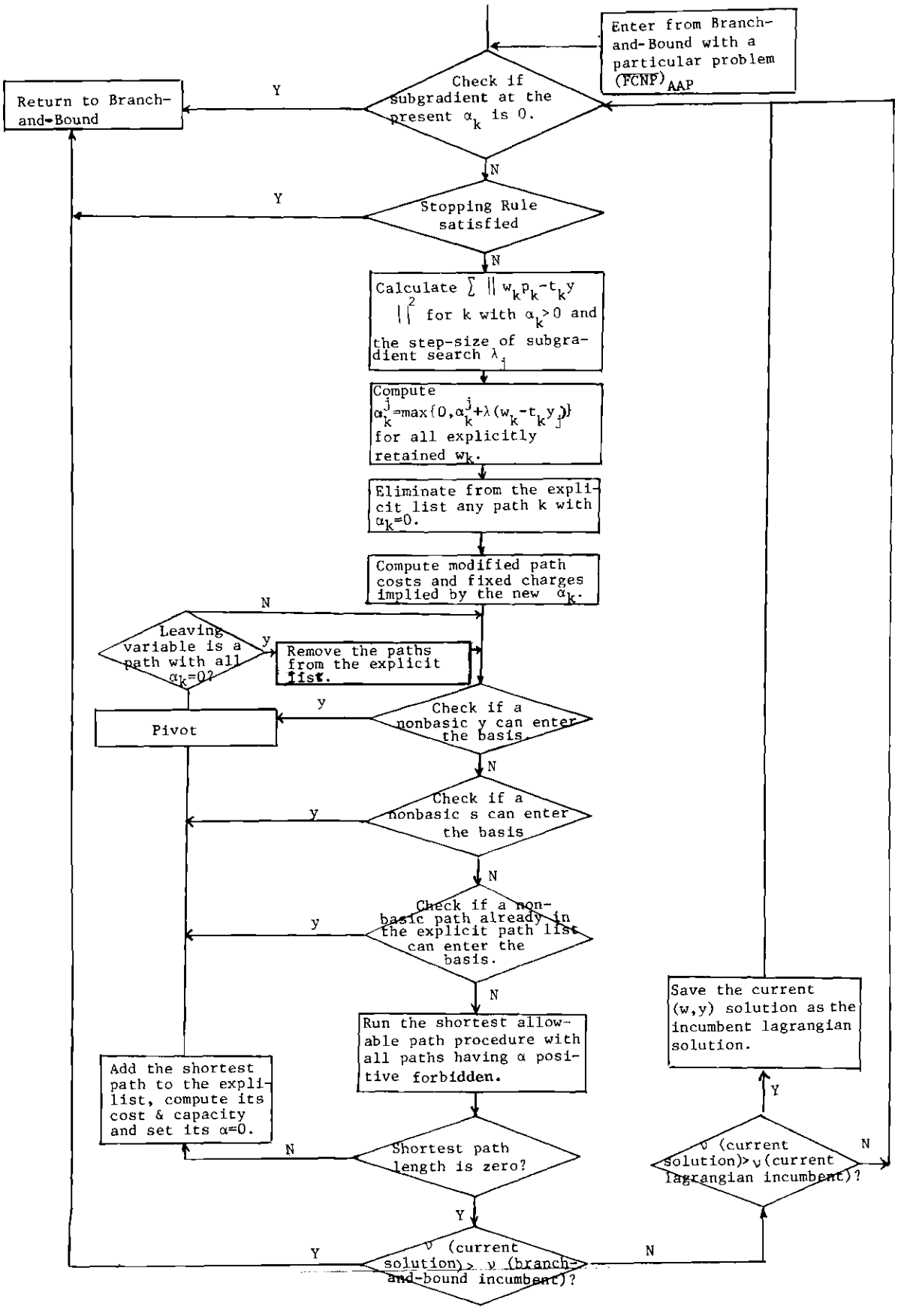


Figure 10. Flow Chart of Lagrangian Relaxation Algorithm for $(\overline{FCNP})_{AAP}$

$$\alpha_k^j = \max \{0, \quad \alpha_k^j + \lambda(w_k - t_k y_j)\}$$

Step 3. If any explicitly known paths now have $\alpha_k = 0$, delete them from the present list.

Step 4. Compute implied path and fixed costs

$$\tilde{\alpha}_k = \alpha_k + \sum_j \alpha_k^j$$
$$\tilde{f}_j = f_j - \sum_k t_k \alpha_k^j$$

for all j and k in the present list.

Step 5. Check if nonbasic y_j can enter the basis. If not, go to next step. Otherwise pivot it into basis, and go to Step 10.

Step 6. Check if a nonbasic slack can enter. If not, go to next step. Otherwise pivot it into basis and go to Step 10.

Step 7. Check if a nonbasic path variable already explicitly known in the present list can enter the basis. If not, go to Step 8. Otherwise, pivot it into basis and go to Step 10.

Step 8. Run the shortest allowable path routine forbidding all paths in the present list having positive dual multipliers α_k. If the resulting path has positive length, go to Step 9. If not, the present linear solution is optimal. Proceed to Step 11.

Step 9. Add the new shortest path to the present list of explicitly retained paths. Calculate its cost α_k and its capacity t_k. Set its $\alpha_k = 0$. Then pivot the new path into the basis and go to Step 10.

Step 10. If the leaving variable of the last pivot was a path with $\alpha_k = 0$, delete it from the present list. Go to Step 5.

Step 11. Compare the optimal linear program solution to the

incumbent lagrangian solution value. If the linear solution is larger save it as a new lagrangian incumbent. If it is as large as the present branch-and-bound incumbent solution, return to branch-and-bound processing and fathom. Otherwise, go to Step 0.

Most elements of the above procedure are direct implementations of the lagrangian concepts developed in the previous section. However, a few explanatory notes are appropriate. First, the present list of paths referred to at Steps 2-4 and 7-10 is the set of paths explicitly retained in computer storage. As noted in Chapter III, all other paths are handled implicitly via the shortest allowable path routine for column generation. Second, negative values of the α_k^j are prohibited at Step 2 to retain feasibility of the α_k. The indicated procedure merely zeros any α component which goes negative as a step is taken. Finally, the need for a lagrangian incumbent solution arises because subgradient search procedures are not monotone in objective function value. The best point found so far must be saved in the incumbent as the search proceeds.

The Complete Branch-and-Bound Procedure

The general outline of a branch-and-bound algorithm for (FCNP) was developed in Figure 1 of Chapter I, and the lagrangian relaxation procedure which will provide fundamental bounds in such an algorithm was developed in the previous section. Before formally stating the final branch-and-bound algorithm of this dissertation, however, a number of decision rules must be discussed.

One such issue is the solution of the first linear relaxation. It would be possible to apply the specialized linear programming procedure developed above for $(\overline{FCNP})_{AP}$ to obtain such a solution. In this case all α_k would be zero. However, it is quite likely that a large number of pivots would be required to obtain this initial solution. An alternative which should be considered is to obtain this first solution via the much more efficient node-arc algorithm. The optimal node-arc solution can then be converted to and arc-path one as discussed at the beginning of Chapter II. One complication arises because the usual resolution of arc flows may not produce a basic path solution (see the discussion in Chapter II). However, this case is easily resolved when the shortest allowable path algorithm of Chapter III is being used to select paths to enter the basis. Paths which would lead to nonbasic solutions can merely be forbidden and the path generation process repeated until one is generated which yields a basic solution.

A second decision rule question in developing a branch-and-bound scheme for FCNP is the question of which candidate problem to pursue at each branch-and-bound iteration. Typically, there are a number of candidate problems pending at any particular selection, and the algorithm must include a rule for selecting the next to explore. The strategies available can range from ones which move to candidate problems quite different from the last investigated to those which seek to make the new problem as similar to the old as possible. The number of candidate problems which must be considered is generally reduced if a wide-ranging rule is selected; however, additional computational effort may be re-

quired in restarting the bounding procedure.

Since the bounds obtained from the $(\overline{FCNP})_{AAP}$ relaxation are expected to be quite good and the procedure for obtaining them quite complex, it appears the strategy of selecting the candidate problem most like the last one is approproate for an algorithm based on bounds from $(\overline{FCNP})_{AAP}$. Specifically, it is proposed that a last-in-first-out rule be applied. If the current candidate problem is not fathomed, the new candidate will be a more restricted version of the current candidate. If the current candidate is fathomed, backtracking is performed until an unexplored candidate is encountered.

One final question in designing a branch-and-bound scheme is the selection of the variable on which to partition problems which cannot be fathomed. There are many ideas for such branching rules, but most depend on the availability of penalties or other conditional information about the solution which might be obtained if a problem was partitioned on a particular variable. Although it might be possible to specialize well-known techniques of this type to the FCNP case, no attempt was made to do so in this dissertation. Thus, the simple rule of choosing the fractional variable y_j with smallest subscript is the one included in the algorithm to be presented.

Implementing these ideas produces the branch-and-bound algorithm depicted in Figure 11. Steps of the procedure are as follows:

Step 0. Read data and initialize branch-and-bound variables. Set $\nu^* = +\infty$, and go to Step 1.

Step 1. Solve the node-arc linear relaxation of the problem and

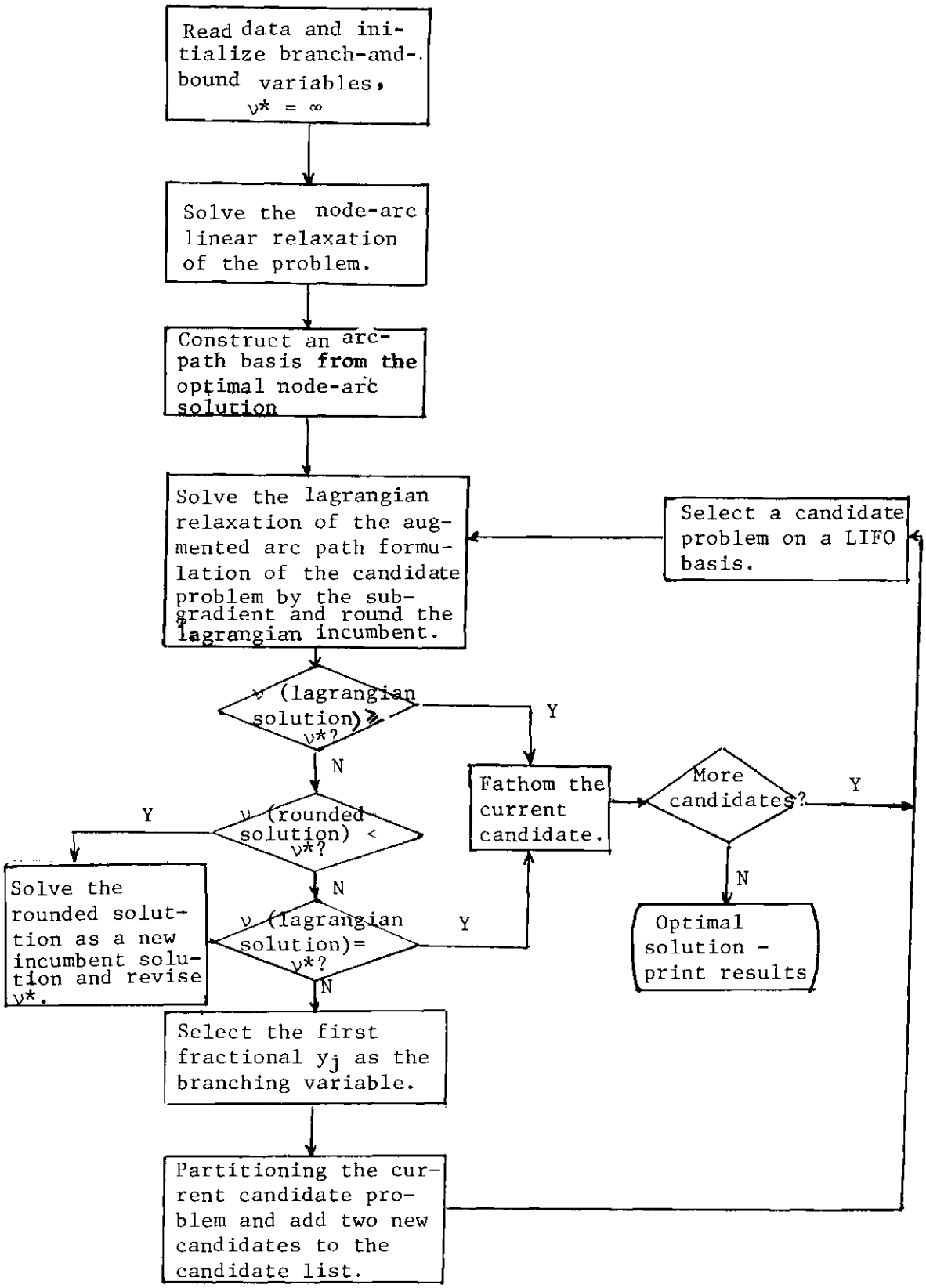


Figure 11. Flow Chart of Branch-and-Bound Algorithm.

go to Step 2.

Step 2. Construct an arc-path basis from the node-arc solution. Then go to Step 3.

Step 3. Solve the lagrangian relaxation of the augmented arc-path form of the current candidate problem by the subgradient search procedure. Also, round the resulting lagrangian incumbent solution. The value of the lagrangian solution is greater than or equal to ν^*, go to Step 7 and fathom. If the rounded solution has value less than ν^* go to Step 4 and save it. Otherwise go to Step 5.

Step 4. Save the rounded optimal solution as a new branch-and-bound incumbent solution. If this new ν^* has value equal to the lagrangian solution value go to Step 7 and fathom. Otherwise, go to Step 5.

Step 5. Select as a branching variable the first free and fraction y_j in the optimal lagrangian solution. Then go to Step 6.

Step 6. Replace the current candidate in the candidate list by two new problems. Both are identical to the current candidate except one has the branching variable fixed at 0 and the other has it fixed at 1. Then go to Step 8.

Step 7. Fathom the current candidate problem from the candidate list because no completion of it can produce a feasible solution with a value less than that of the incumbent solution. If the candidate list is now empty, stop. If an incumbent solution exists, it is an optimal solution for problem, and otherwise, the problem is infeasible. If the candidate list is not empty, proceed to Step 8.

Step 8. Select the next candidate problem to explore on a last-

in-first-out basis, i.e., select the problem most like the last candidate explored. Then go to Step 3.

CHAPTER V

COMPUTATIONAL ANALYSIS

The theoretical developments of Chapter II suggested that augmented arc-path formulations of FCNP's could provide better bounds for branch-and-bound algorithms than the more standard node-arc formulation. Chapters III and IV developed algorithms which solve the augmented arc-path formulation via subgradient search to produce bounds for a branch-and-bound procedure. Here, these theoretical investigations will be supported by empirical evidence on a set of randomly generated test problems. After a discussion of the test problems employed, bounds are compared and solution characteristic are analyzed, including time, number of candidate problems, etc. All of the algorithms presented were implemented in FORTRAN code on the Georgia Institute of Technology's CDC Cyber 76.

Random Test Problem Generation

The problem used in the empirical testing were randomly generated to closely parallel the form of the waste-water system design problems discussed in Jarvis, et al [31]. Such problems have demands for waste-water disposal at various nodes which must be satisfied by building and operating a system of sewer collector lines and sewage treatment plant arcs leading to a super sink. The following five distributions were taken from actual problem data: node demands, fixed

charges on collector arcs, the ratio between fixed charges and variable cost on collector arcs, fixed charges on plant arcs, and the ratio between plant fixed charges and variable costs on plant arcs. Decile points of the five distributions are given in Table 4.

The number of arcs leaving nodes other than the super source and sink were generated using the following distribution: 1 arc leaving in 50% of all cases, 2 arcs leaving in 40% and 3 arcs leaving in 10%. This distribution again corresponds the actual data, although the Jarvis et al problems had parallel arcs and the test problems reported here do not. The number of nodes connected to super source and sink generated at 61% and 17% of total nodes respectively. Again these values correspond to actual waste-water problem data.

Lower bounds on all arcs except demand arcs are 0. Upper bounds equal demands on demand arcs and are otherwise 1 + the maximum flow through the arc. Thus, the only critical arcs (see Chapter II) of the networks are the demand arcs.

FORTRAN code for the test problem generator is included as Appendix B. The following provides a general outline of the generation process.

Step 1. Assign numbers of arcs out of each node between node 2 and last node -2.

Step 2. Link the next to last node to the last node.

Step 3. Assign nodes to be connected to the super-source (node 2 must be included).

Step 4. Connect some node to each non-super-source node.

Table 4. Characteristic of Random Waste-Water Test Problems

Percentage	0	10	20	30	40	50	60	70	80	90	100
Demands	1.	2.	4.	7.	10.	13.	18.	30.	47.	137.	228.
Fixed Charge Costs	20.	63.5	489.5	710.4	930.	1347.2	1617.2	1811.9	2231.	3252.4	4771.9
Fixed Charge/ Variables	10.	22.7	46.6	53.7	113.8	174.8	191.7	195.3	197.	204.2	514.7
Plant Fixed Charge Costs	1062.	1398.	1488.	1596.	3022.	3583.	5278.	19592.	26958.	38146.	74504.
Plant Fixed Chrg./Variable	8.8	8.9	9.0	12.1	13.6	16.2	336.	362.2	382.3	439.6	1072.

Step 5. Generate all other collector arcs, avoiding the duplication.

Step 6. Select nodes to be served by plant arcs. Node last node -1 must be one. Generate plant arcs from each such node to the super sink.

Step 7. Connect the required nodes to the super-source.

Step 8. Assign lower and upper bounds on all arcs.

Step 9. Assign costs on regular arcs, and plant arcs using the given distributions.

Test Performed

Two classes of test problems were generated with above generator, S's (with 21 fixed charge arcs) and M's (with 42 fixed charge arcs) , each problem used a different random number seed. Four test problems of each type were solved by two different procedures. The first is Rardin's [39] fixed charge network algorithm (FCNA) wich uses the node-arc formulation; penalty options in the algorithm were not used. The second algorithm (APNA) is a version of the procedure developed in this research and presented in Chapter IV. FORTRAN code for the APNA procedure is included as Appendix C.

All tests began by solving the linear relaxation of FCNP in node-arc form. FCNA proceeds in node-arc form, while APNA converts to augmented arc-path form for all future calculations.

Computational Results

The principal cause for interest in AAP forms is improvement in

bounds. Table 5 details this aspect of the results. The percent of the LP to IP gap closed by the AAP forms (duals) appears substantial. In fact, the augmented arc-path solution was almost always with 1% of the integer solution. The node-arc formulation had consistently poorer bound performance, and the differences were sometimes dramatic. These results confirm the implications of the small examples in Chapter II and support the value of the AAP form for branch-and-bound procedures.

Table 6 presents the comparison of the run times and numbers of candidate problems, required by the FCNA and APNA algorithms. Solution details (times for particular phases, numbers of LP iterations and subgradient iterations, etc.) are also provided for APNA.

Results for numbers of candidate problems in Table 6 again confirm the value of augmented arc-path bounds in branch-and-bound procedures for FCNP's. The improved bounds of APNA permitted fathoming at an early stage of the enumeration, so that many fewer candidate problems were solved. For larger problems APNA investigated only 1-2% of the candidates explored by the node-arc-based FCNA.

Unfortunately, the solution times reported in Table 6 do not show the same type of improvement. For the smaller problems, APNA typically required several times as much solution time as FCNA. For larger problems the APNA times were competitive, but not significantly improved over those of FCNA.

Review of the breakdown of APNA times and LP iterations in Table 6 helps to isolate the apparent cause of the relatively long solution times. It was often the case that as much as half of the computational

Table 5. Comparison of Bounds

Problems	Node-Arc	Augmented Arc-Path	IP	NA/IP	AAP/IP	IP-NA	IP-AAP
S_1	28062.3	42364.4	42667.7	65.7	99.2	14605.4	303.3
S_2	22016.2	27617.7	32333.7	68.0	85.0	10317.5	4716.0
S_3	93995.3	100910.	101003.	93.0	99.9	7007.7	93.
S_4	443951.1	457358.	457943.	96.9	99.8	13991.9	585.
M_1	101374.	113630.	114710.	88.3	99.0	13336.	1080.
M_2	150263.5	161496.	162942.	92.2	99.1	12678.5	1446.
M_3	576079.7	596672.	598014.1	96.3	99.7	21934.4	1342.1
M_4	1091293.7	1117010.	1117830.	97.6	99.9	26536.3	820.

Table 6. Comparison of Run Time, Number of Candidates Problems, and Number of LP and Subgradient Iterations

PROBLEMS	Run Time(SEC)		Numbers of Candidate Problems		First Network Time (FCNA)	APNA Details			
						Arc-Path Basis Construction Time(No. of LP Iterations)	First AAP Solution Time (No. of LP iterations)	All other AAP Calculation Time (No. of LP iterations)	Number of Subgradient iterations (APNA)
	FCNA	APNA	FCNA	APNA					
S_1	4.1	33.8	111	13	.038	0 (13)	15.8 (243)	17.8 (190)	241
S_2	2.0	178.2	73	21	.04	0 (13)	28.3 (655)	149.6 (3674)	551
S_3	2.6	14.0	99	9	.03	0 (13)	6.3 (102)	7.4 (109)	156
S_4	1.2	19.3	53	11	.036	0 (13)	10.4 (228)	8.7 (198)	207
Average	2.4	61.3	84	13.5	.036	0 (13)	15.2 (322)	45.8 (1042.7)	290
M_1	140.5	120.4	1777	17	.082	1 (25)	67.9 (538)	51.2 (265)	374
M_2	33.6	145.5	619	21	.078	1 (25)	43.7 (396)	100.2 (746)	383
M_3	352.9	158.3	4285	17	.09	1 (25)	72.5 (681)	84.3 (778)	287
M_4	50.6	55.6	871	19	.08	1 (25)	11.2 (85)	43.2 (276)	280
Average	144.2	119.5	1888	18.5	.082	1 (25)	48.8 (425)	69.7 (516.2)	331

effort to solve APNA was consumed in obtaining the first augmented arc-path solution. That large segment of APNA time was required because of relatively slow convergence of the subgradient search procedure used to solve the AAP form. Very substantial savings would probably result from substitution of some improved scheme that reached the augmented arc-path solution more rapidly.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

The primary objective of the research reported in this dissertation was to investigate theoretical and algorithmic issues in the use of arc-path forms (on their extensions) to solve general fixed charge network problems. Properties of fixed charge network formulations were developed in Chapter II, algorithmic developments are presented in Chapters III and IV, and computational experience is reported in Chapter V. The principal results can be summarized as follows:

1. Analysis of Arc-Path Formulations to the Fixed Charge Network Problems. Three formulations of FCNP were presented and relations between those formulations were demonstrated in terms of the bound improvements.

2. Characterization of Basis Inverse Structure of the Arc-Path Linear Relaxation. The linear relaxation of FCNP in arc-path form was shown to have bases with an identifiable and exploitable structure. The structure led in Chapter V to the statement of a specialized simplex algorithm for that relaxation which explicitly stores only a small part of the basic inverse matrix.

3. Investigation of Gaps Between the Various Formulations of Fixed Charge Network Problems. In Chapter II it was demonstrated that arc-

path formulations of (FCNP) can yield greater lower bounds than node-arc approaches. Computational experience in Chapter V contains empirical verification of those results for problems of practical size.

4. Development of Shortest Allowable Path Procedure. The methods used by Tomlin [44] and Jarvis [30] to solve arc-path formulations can be viewed as revised simplex procedures with a special column generation scheme which involves the solution of a shortest path problem. An efficient shortest allowable path procedure was developed in Chapter III that permits extension of the Tomlin/Jarvis approach to many problems where all components of the adjusted cost on paths cannot be allocated to arcs. Empirical results which were presented in Chapter II suggest that the algorithm would require no more than twice the time of Dijkstra's procedure for the ordinary shortest paths in acyclic networks.

5. Development of a Branch-and Bound Procedure for (FCNP) Exploiting the Augmented Arc-Path Formulation. In Chapter IV the above results were combined with known techniques to produce a complete branch-and-bound algorithm which solves FCNP. Empirical evidence presented in Chapter V suggests this procefure greatly reduces number of candidate problems solved by providing better lower bounds than the other formulations. However, computation times are generally no better than competitive procedures.

Taken together the above results do appear to at least suggest that arc-path approaches are worthy of more serious attention as the center of branch-and-bound procedures for FCNP's. In particular the empirical results of Chapter V strongly suggest that the bounds obtained

from such procedures can be of very great assistance in limiting the amount of enumeration required. However, the solution times reported in Chapter V were not satisfactory. The principal problem appears to be rather slow convergence of the sub-gradient procedure that was used to solve the augmented arc-path relaxation of FCNP. Thus it is strongly recommended that further research be directed to the identification of an alternative decomposition approach for solving the augmented relaxation. Possible avenues of research would include (i) searches for structure in the master problem implied by the path capacity constraints that could be exploited to make more standard decomposition procedures feasible for the augmented arc-path relaxation, and (ii) attempts to reverse the roles of the arc-path relaxation and the path capacity constraints in the augmented formulation, so that the latter become the subproblem and yet the structure of the former can be exploited in solving master problems in a decomposition.

As a further extension, it is suggested that research be directed to the use of penalty procedures in combination with the augmented arc-path formulation of FCNP. Rardin and Unger's [40] results for the node-arc case suggest that such penalties are rather powerful in FCNP's. The penalties would probably be somewhat more difficult to compute in arc-path formulations because the initial simplex tableau of the problem is not readily available. However, the success in node-arc algorithms suggests that research on penalties in the arc-path case might be quite productive.

APPENDIX A

RANDOM NETWORK GENERATOR FOR SHORTEST FORBIDDEN PATHS

```
      PROGRAM NETGEN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C-----TO GENERATE RANDOM ACYCLIC NETWORK PROBLEMS
C
      COMMON IBEG(1000),IEND(1000),IPTH(10000),FCOST(1000),
     *  VCOST(1000),
     *UBND(1000),LBND(1000),NUMELM(100),NUMNODE,NUMARC,
     *  NUMPTH,
     *NUMOAR(300),MARK(32,300),NUM1(100),IFIRARC(300),TIME,
     *  ATINE
      DIMENSION IVCOST(1000),SUMARC(100),EAGLE(300),
     *POOL(300),PROW(300)
      INTEGER UBND,SUMARC
C-----SET UP
      READ(5,*)NUMNODE,NUMARC,NUMPTH,COST,P
      WRITE(6,5) NUMNODE,COST,P
    5 FORMAT(1H0,1X,"NUMNODE=",I5/1X,"COST=",F10.4/1X,
     *"P    =",F10.4/1X)
      CALL RANSET(P)
      DO 10 I=1,NUMARC
      FCOST(I)=10.
      UBND(I)=20
   10 LBND(I)=0
      DO 4 I=1,32
      DO 4 J=1,NUMNODE
    4 MARK(I,J)=0
C-----ASSIGN AN ARC TO EACH COLUMN
      NNA=0
      NUM=NUMNODE-1
      DO 40 J=2,NUMNODE
      Z=RANF(P)
      L=J-1
C-----COMPUTE CELL PROBABILITIES
      DO 35 I=1,L
      POOL(J)=1.*(I*(I+1))/(J*(J-1))
      IF(Z.LE.POOL(J)) GO TO 31
   35 CONTINUE
   31 CALL MKBUMP(I,J)
      NNA=NNA+1
C     PRINT*,I,J,Z,POOL(J),NNA
   40 CONTINUE
C-----ASSIGN ARCS TO ROW
      DO 24 I=1,NUM
      Z=RANF(P)
      K=I+1
      NU=NUMNODE
C-----COMPUTE CELL PROBABILITIES
      DO 26 J=K,NUMNODE
      PROW(I)=1.*((NU-J)*(NU-J+1))/((NU-I)*(NU-I+1))
      IF(Z.GE.PROW(I)) GO TO 21
   26 CONTINUE
```

```
   21   CALL MKBUMP(I,J)
        NNA=NNA+1
C       PRINT*,I,J,Z,PROW(I),NNA
   24   CONTINUE
C-----ASSIGN THE REMAINING ARC TO ROW
C-----PICK A ROW AT RANDOM
   39   Z=RANF(P)
        I=1+RANF(P)*(NUMNODE-1)
        M=I+1
        DO 41 J=M,NUMNODE
        PROW(I)=1.*((NU-J)*(NU-J+1))/((NU-I)*(NU-I+1))
        IF(Z.GE.PROW(I)) GO TO 43
   41   CONTINUE
   43   IF(MKGET(I,J).GE.15) GO TO 39
        CALL MKBUMP(I,J)
        NNA=NNA+1
C       PRINT*,I,J,Z,PROW(I),NNA
        IF(NNA.EQ.NUMARC) GO TO 45
        GO TO 39
C-----ASSIGN NA. AND VCOST TO CHECKED ARCS
C
   45   NA=1
        DO 50 I=1,NUM
        DO 50 J=2,NUMNODE
        IF(MKGET(I,J).EQ.0) GO TO 50
        KK=MKGET(I,J)
        DO 52 NK=1,KK
        IBEG(NA)=I
        IEND(NA)=J
        IVCOST(NA)=RANF(P)*COST
        VCOST(NA)=IVCOST(NA)
   52   NA=NA+1
   50   CONTINUE
C-----DETERMINE NUMOARC(I) & IFIRARC(I)
        NUMOAR(1)=0
        DO 60 I=1,NUMARC
        IF(I.GT.1) GO TO 55
        NI=IBEG(I)
        IFIRARC(NI)=I
   54   NUMOAR(NI)=NUMOAR(NI)+1
        GO TO 60
   55   IF(NI.EQ.IBEG(I)) GO TO 54
        NI=IBEG(I)
C       PRINT*,NI
        IFIRARC(NI)=I
        NUMOAR(NI)=0
        GO TO 54
   60   CONTINUE
        IF(NUMPTH.EQ.0) GO TO 81
C-----GENERATE NO. OF FORBIDDEN PATHS
```

```
      L=0
      LASTP=0
      DO 80 NP=1,NUMPTH
 74   NUMELM(NP)=0
      SUMARC(NP)=0
      ND=1
  75  MAXK=NUMOAR(ND)
      NA=IFIRARC(ND)+RANF(P)*NUMOAR(ND)
C
      L=L+1
      IF(L.GT.LOOP-NUMNODE) STOP 777
      IPTH(L)=NA
C     PRINT*,ND,MAXK,L,NA
      ND=IEND(NA)
      NUMELM(NP)=NUMELM(NP)+1
      SUMARC(NP)=SUMARC(NP)+NA
      IF(ND.EQ.NUMNODE) GO TO 76
      GO TO 75
C-----CHECK WHETHER PATHS ARE DUPLICATED
  76  IFIRP=LASTP+1
      NUM1(NP)=IFIRP
      LASTP=IFIRP+NUMELM(NP)-1
      IF(NP.EQ.1) GO TO 80
      KMAX=NP-1
      DO 77 K=1,KMAX
      IF(SUMARC(NP).EQ.SUMARC(K).AND.NUMELM(NP).EQ.
     *NUMELM(K)) GO TO 78
C     PRINT*,K,NP,SUMARC(K),SUMARC(NP),NUMELM(K),NUMELM(
C    *NP)
      GO TO 77
  78  NJ=NUM1(K)
      KELM=NUMELM(K)
      NJJ=NUM1(NP)
C-----CHECK ELEMENTS OF PATHS
      DO 79 J=1,KELM
      IF(IPTH(NJ).EQ.IPTH(NJJ)) GO TO 72
      GO TO 77
C72   PRINT*,NJ,NJJ
 72   NJ=NJ+1
      NJJ=NJJ+1
  79  CONTINUE
      GO TO 73
  77  CONTINUE
      GO TO 80
C-----RESET IPTH(L)
  73  L=L-NUMELM(NP)
      LASTP=LASTP-NUMELM(NP)
      IFIRP=IFIRP-NUMELM(NP)+1
      GO TO 74
  80  CONTINUE
```

```
   81   CONTINUE
        WRITE(6,6) NUMARC
    6   FORMAT(1H0,1X,"NUMARC=",I5/1X)
        WRITE(6,100)
        WRITE(6,91)(NA,IBEG(NA),IEND(NA),LBND(NA),
       *UBND(NA),VCOST(NA),FCOST(NA),NA=1,NUMARC)
        WRITE(6,7) NUMPTH
    7   FORMAT(1H0,1X,"NUMPTH=",I5/1X)
        WRITE(6,92)
        LASTP=0
        IF(NUMPTH.EQ.0) GO TO 910
        DO 90 NP=1,NUMPTH
        IFIRP=LASTP+1
        LASTP=IFIRP+NUMELM(NP)-1
   90   WRITE(6,93) NP, NUMELM(NP),(IPTH(I),I=IFIRP,LASTP)
  910   CONTINUE
        CALL SHORT(NUMNEW,EAGLE)
   11   FORMAT(1H0,1X,"TIME=",F10.4/1X,"ATIME=",F10.4/1X)
        WRITE(6,11)TIME,ATIME
C       FORBIDDEN PATH DATA
        IF(NUMPTH.EQ.0) GO TO 911
C       PRINT SHORTEST PATH
  911   CONTINUE
        WRITE(6,94)
        IP=NUM1(NUMNEW)
        IPP=IP+NUMELM(NUMNEW)-1
        WRITE(6,93)NUMNEW, NUMELM(NUMNEW),(IPTH(I),I=IP,IPP)
        WRITE(6,96) EAGLE(NUMNODE)
  100   FORMAT(1H0,20X,"ARC DATA "//3X," NO.   BEG   END",
       *"     LBND      UBND        VCOST        FCOST   "/1X)
   91   FORMAT(1H ,I5,2I6,2I9,2F12.4)
   92   FORMAT(1H0,20X," PATH DATA"//3X,"NO. NO. OF ARC",
       *"     PATH ELEMENTS"/1X)
   93   FORMAT(1H ,I5,4X,I5,10I6)
   94   FORMAT(1H0,20X,"SHORTEST PATH"//3X,"NO. NO. OF ARC",
       *"     PATH ELEMENTS"/1X)
   96   FORMAT(1H0,20X," SHORTEST DISTANCE"//20X,F12.4/1X)
        STOP
        END
```

```
        SUBROUTINE MKBUMP(I,J)
C-----TO HANDLE BIG (I,J) MATRIX
        COMMON IBEG(1000),IEND(1000),IPTH(10000),FCOST(1000),
       *VCOST(1000),UBND(1000),LBND(1000),NUMELM(100),NUMNODE,
       *NUMPTH,NUMOAR(300),MARK(32,300),NUM1(100),IFIRARC(300)
       *   ,
```

```
      *TIME,ATIME
       IWD=I/16
       IPOS=I-16*IWD
       MARK(IWD+1,J)=MARK(IWD+1,J)+16**IPOS
       RETURN
       END



       FUNCTION MKGET(I,J)
C-----TO HANDLE BIG (I,J) MATRIX
       COMMON IBEG(1000),IEND(1000),IPTH(10000),FCOST(1000),
      *VCOST(1000),UBND(1000),LBND(1000),NUMELM(100),NUMNODE,
      *NUMPTH,NUMOAF(300),MARK(32,300),NUM1(100),IFIRARC(300)
      *   ,
      *TIME,ATIME
       IWD=I/16
       IPOS=I-16*IWD
       IWORK=MARK(IWD+1,J)/16**IPOS
       MKGET=IWORK-(IWORK/16)*16
       RETURN
       END
```

APPENDIX B

RANDOM WASTE-WATER NETWORK GENERATOR

```
      PROGRAM RNDSEW(TAPE2,INPUT,OUTPUT,TAPE5=INPUT,TAPE6=
     *  OUTPUT)
      DIMENSION IX(500),IY(500),FCOST(500),VCOST(500),
     *  IUBND(500),
     *LBND(500),LST1(100),LST2(100),NODREM(100),IREACH(100),
     *NAM(5),IPHASE(5)
      DATA IPHASE,ENUF,STUP/0,0,0,0,0.0,1.0E46/
      LP1=3
      LP2=4
      INFIN=1000000
      LUD=2
      REWIND LUD
      JA=0
      P=12345.
C-----READ PARAMETERS
      READ*,NUM1,NUM2,NUM3,NUMSRC,NUMSNK,NUMPASS
      READ(5,100)NAM
100   FORMAT(5A4)
      DO 150 I=1,NUMPASS
150   R=RANF(P)
      NUMFCA=NUMSNK+NUM1+2*NUM2+3*NUM3
      NUMARC=NUMFCA+NUMSRC+1
      NUMNOD=NUM1+NUM2+NUM3+3
C-----INITIALIZE
      DO 190 J=1,NUMARC
      LBND(J)=0
      IUBND(J)=INFIN
      FCOST(J)=0.
190   VCOST(J)=0.
C-----ESTABLISH NO. ARCS OUT OF EACH NODE
      CALL SHUFFL(LST1,P,2,NUMNOD-2)
      NODREM(NUMNOD)=0
      NODREM(NUMNOD-1)=0
      NODREM(1)=0
      DO 211 K=1,NUM1
      ND=LST1(K)
210   NODREM(ND)=1
      KMIN=NUM1+1
      KMAX=NUM1+NUM2
      DO 221 K=KMIN,KMAX
      ND=LST1(K)
220   NODREM(ND)=2
      KMIN=KMAX+1
      KMAX=KMAX+NUM3
      DO 231 K=KMIN,KMAX
      ND=LST1(K)
230   NODREM(ND)=3
C     PRINT*,"NODREM=",(NODREM(I),"/",I=1,NUMNOD)
C-----HANDLE NEXT TO LAST NODE
      ND=LST1(1)
```

```
      NODREM(ND)=NODREM(NUMNOD-2)
      NODREM(NUMNOD-2)=1
      IF(NODREM(NUMNOD-3).LE.2) GO TO 300
      ND=LST1(2)
      NRSAV=NODREM(NUMNOD-3)
      NODREM(ND)=NODREM(NUMNOD-3)
      NODREM(NUMNOD-3)=NRSAV
C-----ESTABLISH NODES CONNECTED TO SUPER SOURCE
300   CALL SHUFFL(LST2,P,3,NUMNOD-1)
C-----CONNECT THOSE NOT CONNECTED TO SUPER SOURCE
      KMIN=NUMSRC
      KMAX=NUMNOD-3
C-----CONNECT NON SUPER-SOURCE NODES
      DO 390 K=KMIN,KMAX
      KSAV=KMIN
      DO 330 L=KMIN,KMAX
      IF(LST2(L).GE.LST2(KSAV)) GO TO 330
      KSAV=L
330   CONTINUE
      ND=LST2(KSAV)
      LST2(KSAV)=INFIN
340   MD=2+RANF(P)*(ND-2)
      IF(NODREM(MD).LE.0) GO TO 340
      NODREM(MD)=NODREM(MD)-1
      JA=JA+1
      IX(JA)=MD
      IY(JA)=ND
390   CONTINUE
C-----FINISH REGULAR ARCS
      MMAX=NUMNOD-2
      DO 450 MD=2,MMAX
430   IF(NODREM(MD).EQ.0) GO TO 450
      JA=JA+1
440   ND=1+MD+RANF(P)*(NUMNOD-MD-1)
      IX(JA)=MD
      IY(JA)=ND
C-----CHECK DUPLICATION
      IF(M2(JA,IX,IY).NE.0) GO TO 440
      NODREM(MD)=NODREM(MD)-1
      GO TO 430
450   CONTINUE
C-----SELECT PLANT NODES
      CALL SHUFFL(LST1,P,2,NUMNOD-2)
C-----SET NODE NUMNOD-1
      JA=JA+1
      IX(JA)=NUMNOD-1
      IY(JA)=NUMNOD
C-----DO OTHERS
      KMAX=NUMSNK-1
      DO 550 K=1,KMAX
```

```
      MD=LST1(K)
      JA=JA+1
      IX(JA)=MD
      IY(JA)=NUMNOD
550   CONTINUE
C-----CONNECT NODES TO SUPER-SOURCE
C-----NODE 2
      JA=JA+1
      IX(JA)=1
      IY(JA)=2
      LBND(JA)=RVAL(1,P)
      IUBND(JA)=LBND(JA)
C-----OTHERS
      KMAX=NUMSRC-1
      DO 590 K=1,KMAX
      ND=LST2(K)
      JA=JA+1
      IX(JA)=1
      IY(JA)=ND
      LBND(JA)=RVAL(1,P)
      IUBND(JA)=LBND(JA)
590   CONTINUE
C-----SET RETURN ARC
      JA=JA+1
      IX(JA)=NUMNOD
      IY(JA)=1
C-----LABEL TO GET CAPACITIES
      DO 630 MD=1,NUMNOD
630   IREACH(MD)=0
      JMIN=NUMFCA+1
      JMAX=NUMARC-1
      DO 640 J=JMIN,JMAX
      ND=IY(J)
      IREACH(ND)=LBND(J)
640   CONTINUE
      MMAX=NUMNOD-1
      DO 690 MD=2,MMAX
      DO 680 JA=1,NUMARC
      IF(IX(JA).NE.MD) GO TO 680
C-----INCREASE LABEL AND SET CAPACITY
      ND=IY(JA)
      IREACH(ND)=IREACH(ND)+IREACH(MD)
      IUBND(JA)=IREACH(MD)+1
680   CONTINUE
690   CONTINUE
C-----SET COSTS ON REGULAR ARCS
      JMAX=NUMFCA-NUMSNK
      DO 730 JA=1,JMAX
      FCOST(JA)=RVAL(2,P)
      VCOST(JA)=FCOST(JA)/RVAL(3,P)
```

```
      VCOST(JA)=INT(VCOST(JA)*100.+.5)/100.
      FCOST(JA)=FCOST(JA)/IUBND(JA)
730   FCOST(JA)=INT(FCOST(JA)*100.+.5)/100.                          85
C-----SET COSTS ON PLANT ARCS
      JMIN=JMAX+1
      DO 750 JA=JMIN,NUMFCA
      FCOST(JA)=RVAL(4,P)
      VCOST(JA)=FCOST(JA)/RVAL(5,P)
      VCOST(JA)=INT(VCOST(JA)*100.+.5)/100.
      FCOST(JA)=FCOST(JA)/IUBND(JA)
750   FCOST(JA)=INT(FCOST(JA)*100.+.5)/100.
C-----WRITE
      WRITE(LUD)NAM,NUMARC,NUMNOD,NUMFCA,STUP
      PRINT*,NAM,NUMARC,NUMNOD,NUMFCA,LP1,LP2,IPHASE,ENUF,
     *  STUP
      DO 850 JA=1,NUMFCA
      WRITE(LUD)IX(JA),IY(JA),LBND(JA),IUBND(JA),
     *VCOST(JA),FCOST(JA)
      PRINT*,IX(JA),IY(JA),LBND(JA),IUBND(JA),
     * VCOST(JA),FCOST(JA)
850   CONTINUE
      JMIN=NUMFCA+1
      DO 870 JA=JMIN,NUMARC
      WRITE(LUD)IX(JA),IY(JA),LBND(JA),IUBND(JA),
     *VCOST(JA)
      PRINT*,IX(JA),IY(JA),LBND(JA),IUBND(JA),VCOST(JA)
870   CONTINUE
      END FILE LUD
      REWIND LUD
      STOP
      END




      SUBROUTINE SHUFFL(LSTNOD,P,N1,N2)
C-----SHUFFLE NODE NUMBERS
      DIMENSION LSTNOD(1)
      IMAX=N2-N1+1
      DO 100 I=1,IMAX
100   LSTNOD(I)=I+N1-1
      KMAX=20*N2
      DO 200 K=1,KMAX
      I1=1+RANF(P)*IMAX
      I2=1+RANF(P)*IMAX
      ND=LSTNOD(I1)
      LSTNOD(I1)=LSTNOD(I2)
      LSTNOD(I2)=ND
```

```
200     CONTINUE
C       PRINT*,"LSTNOD=",(LSTNOD(I),"#",I=1,IMAX)
        RETURN
        END
```



```
        FUNCTION M2(JA,IX,IY)
C-----CHECK FOR DUPLICATE OF PREVIOUS 2 ARCS
        DIMENSION IX(1),IY(1)
        M2=0
        IF(JA.LE.1) GO TO 900
        JMIN=1
        JMAX=JA-1
        DO 200 J=JMIN,JMAX
        IF(IX(J).EQ.IX(JA).AND.IY(J).EQ.IY(JA)) M2=1
200     CONTINUE
C       PRINT*,"DUPLICATION CHECK=",JA,IX(JA),IY(JA),M2
900     RETURN
        END
```

```
        FUNCTION RVAL(J,P)
C-----GENERATE RANDOM VAL OF (J=1) DEMANDS
C       (J=2) REGULAR FCOST (J=3) REGULAR FCOST/VCOST
C       (J=4) PLANT FCOST (J=5) PLANT FCOST/VCOST
        DIMENSION DISTR(11,5)
        DATA DISTR/
       *1.0,2.0,4.0,7.0,10.0,13.0,18.0,30.0,47.0,137.0,228.0,
       *20.0,63.5,469.5,713.4,938.,1347.2,1617.2,1811.9,
       *  2231.,3252.4,4771.9,
       *10.0,22.7,46.6,53.7,113.8,174.8,191.7,195.3,197.0,
       *  204.2,514.7,
       *1062.,1396.,1488.,1596.,3022.,3583.,5276.,19592.,
       *  26958.,38146.,74514.,
       *8.8,8.9,9.0,12.1,13.6,16.2,336.,362.2,382.3,439.6,
       *  1072./
        R=RANF(P)
        I=1+10.*R
        RVAL=DISTR(I,J)*(1.-R)+DISTR(I+1,J)*R
        RVAL=INT(RVAL+.5)
C       PRINT*,"RVAL=",RVAL
        RETURN
        END
```

APPENDIX C

ARC-PATH NETWORK ALGORITHM

```
      PROGRAM APNA(TAPE1,INPUT,OUTPUT,TAPE5=INPUT,TAPE6=
     *  OUTPUT)
C-----PROGRAM SOLVE FIXED CHARGE NETWORK PROBLEMS IN
C     ARC-PATH FORM WITH THE BRANCH-AND BOUND BY
C       SUBGRADIENT METHOD
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *  ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *  ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *  OBJ,ITLP,
     *QL(75),PINV(75,75),ALPA(500),NUMGUB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,1500)
      INTEGER UBND
      DIMENSION XX(500),TFCOST(500),REF(1),IFIXST(500),
     *  NAMPRB(5)
C
C-----SECTION TO INITIALIZE-----
C
C101  FORMAT(1X,I3)
C     READ(5,101) ILVLPR
C-----READ PROBLEM FROM NODE-ARC FORMS
      ILVLPR=2
      PRINT*,"ENTER LEVEL OF PRINT 1=LEAST INFORMATION,
     *2=INCLUDE LP,4=INCLUDE LP+LAGRANGEAN,5=TABLX"
      READ*,LP
      REWIND 1
      READ(1) NUMNODE,NUMARC,NUMFCA,NUMGUB,NAMPRB
810   FORMAT(1H1,"NAME=",5A4
     *  /1X,"NO. NODES=",I6
     *  /1X,"NO. ARCS=",I6
     *  /1X,"NO. FIX CHARGE ARCS=",I6
     *  /1X,"NO. GUB ARCS=",I6)
815   FORMAT(1H0,20X,"ARC DATA"//3X,
     *  "NO.  BEG  END    LBND    UBND      VCOST",
     *  "       FCOST         X         XINCUM  "/1X)
825   FORMAT(1H0,20X,"GUB DATA"//3X,
     *  "NO.   BEG    NO. ELEMENT")
820   FORMAT(1H ,I4,2I5,2I8,4F12.4)
```

```
830    FORMAT(1H ,I5,2I6)
840    FORMAT(1H1,"ICRITER=",I6
     *  /1X,"LEVEL OF PRINT",I5
     *  /1X,"SCALING=",F10.4
     *  /1X,"PATH CAPACITY SCALING=",F10.4
     *  /1X,"MIN STEP SIZE=",F10.4)
       DO 2 J=1,NUMARC
       READ(1) IBEG(J),IEND(J),LBND(J),UBND(J),VCOST(J),
     *FCOST(J),X(J)
       IF(LBND(J).GT.TOL(3))UBND(J)=LBND(J)
2      CONTINUE
       IF(NUMGUB.EQ.0) GO TO 4
       READ(1) (IYGUB(I),IYELM(I),I=1,NUMGUB)
C-----INITIALIZE PROGRAM LIMITS
     4 MAXNOD=150
       MAXARC=500
       MAXFCA=100
       INFIN=10000000
       LUPR=6
       LUCD=5
       MAXBB=1500
       VINCUM=INFIN
C-----TOL(1) FOR TEST OF SIGN OF TABLE COEFFICIENTS
C      TOL(2) FOR TEST OF ZC, TOL(3) FOR TEST OF INTEGRALITY
C      TOL(4) FOR IMPROVEMENT TEST, TOL(5) FOR FIXED CHARGE
C        ARC TEST
       TOL(1)=1.E-5
       TOL(2)=1.E-5
       TOL(3)=1.E-5
       TOL(4)=1.E-3
       TOL(5)=1.E-7
       IWENTER=0
       IYENTER=0
       ISENTER=0
C-----SCAL= SCALE FACTOR,PCSCAL=PATH CAPACITY SCALE FACTOR
C      DMIN=MINIMUM STEP SIZE
       PRINT*,"ENTER ICRITER,SCAL,DMIN,PCSCAL  "
       READ*,ICRITER,SCAL,DMIN,PCSCAL
C      ICRITER LOGIC
       INCPIT=1
       XINFIN=INFIN
C-----ALL BETA(I)=0.
       DO 3 I=1,NUMGUB
3      BETA(I)=0.
C-----DETERMINE NUMOAR(I) & IFIRARC(I)
       NUMOAR(1)=0
       DO 20 I=1,NUMARC
       IF(I.GT.1) GO TO 25
       NI=IBEG(I)
       IFIRARC(NI)=I
```

```
24      NUMOAR(NI)=NUMOAR(NI)+1
        GO TO 23
25      IF(NI.EQ.IBEG(I)) GO TO 24
        NI=IBEG(I)
        IFIRARC(NI)=I
        NUMOAR(NI)=0
        GO TO 24
20      CONTINUE
        IF(LP.GE.4)
       *PRINT*,"NUMOAR INF=",(I,IFIRARC(I),NUMOAR(I),I=1,
       *  NUMNODE)
C-----INITIALIZE B-B SYSTEMS
C       LLC/LOC OPTION 1=LLC,2=LOC
        ILOOP=1
C       SEQUENCE OPTION 1=BOUND,2=AUXILIARY
        IISQOP=1
C       PROBLEM 0-1 OR GENERAL
        IISO1=1
C       LENGTH OF DECISION VECTOR AND SOLUTION VECTOR
C       MAXIMUM NUMBER OF NODE SPACES
C       DEVICE NUMBER OF PRINTER
C       SUBOPTIMALITY PERCENTAGE OF SOLUTION
        XPTSOP=0.
C       STARTING UPPER BOUND
        XCSTIN=INFIN
        CALL BBINIT(NAMPRB,ILOOP,IISQOP,IISO1,NUMARC,NUMARC,
       *  MAXBB,
       *  2*NUMARC,LUPR,ILVLPR,XPTSOP,XCSTIN)
C
C-----SECTION TO SELECT CANDIDATE-----
C
C       RETURN STSTUS (1=SELECTION IS AN EXTENSION OF THE
C         LAST ONE,
C       2=SELECTION IS NOT AN EXTENSION OF THE LAST ONE)
202     CALL BBSELC(BND,AUX,IFIXST,REF,REF,IRTNST)
C
C-----SECTION TO CONSTRUCT ARC-PATH FORM BASIS
C
        ISTOP=0
        TINFIN=INFIN
        ITLP=0
        DO 204 I=1,NUMARC
        Y(I)=0.
        XX(I)=X(I)
204     XLAGR(I)=X(I)
        APSTAT=SECOND(P)
        CALL APBASIS
        ITLPAP=ITLP
        APFIN=SECOND(P)
        APTIME=INT((APFIN-APSTAT)*1000+.5)/1000
```

```
      ITLAGR=0
      VSTOP=VLAST
      ISFIR=0
      VLAGR=VLAST
      OBJ=VLAST
203   DO 201 I=1,NUMARC
201   TFCOST(I)=FCOST(I)
      FFIX=0.
C-----OBTAIN INCUMBENT WITH ROUND SOLUTION
      CALL RNDSOL(VRND)
205   CALL BBFEAS(VRND,X,XINCUM,IRTNST)
      IF(VINCUM.GT.VRND)VINCUM=VRND
      PRINT*,"VLAST=",VLAST,"VRND=",VRND
      GO TO (1,202,900),IRTNST
C
C-----SECTION TO SOLVE LAGRANGEAN RELAXATION BY SUBGRADIENT
C
C-----COMPUTE INFEASIBILITY OF PATH CAPACITY CONSTRAINT
1     DEL=0.
      ITLAGR=ITLAGR+1
      IF(LP.GE.2)
     *PRINT*,"***********BEGIN LAGR ITER ",ITLAGR
C     IF(ITLAGR.LE.300) GO TO 333
C     STOP 2
      ITLPL=0
      IFEAS=1
      ICOMP=1
      IF(NUMPTH.EQ.0) GO TO 7
      DO 334 I=1,NUMPTH
334   DTIL(I)=0.
      IF(ISFIR.EQ.1) GO TO 45
      DO 6 IW=1,NUMPTH
      NW=IP1BW(IW)
      NUM=NUM1(NW)
      NUMEL=NUMEL(NW)+NUM-1
      DO 5 I=NUM,NUMEL
      NA=IPTH(I)
      IF(IYBSTS(NA).GE.4) GO TO 5
C     PATH CAPACITY SCALE FACTOR
      TDEL=(W(NW)-T(NW)*Y(NA))*PCSCAL/T(NW)
      DEL=DEL+TDEL**2
C     PRINT*,"I NW W(NW) T(NW) Y(NA) ",I,NW,W(NW),T(NW),
C       Y(NA)
C     PRINT*,"TDEL ",TDEL
      IF(TDEL.GT.TOL(3))IFEAS=0
      IF(TDEL*ALPA(I).GT.TOL(3).OR.TDEL*ALPA(I).LT.-TOL(3))
     *  ICOMP=0
5     CONTINUE
6     CONTINUE
C-----COMPUTE INFEASIBILITY OF SUBCONSTRAINT
```

```
7       CONTINUE
        EPSI=0.
        IF(NUMGUB.EQ.0) GO TO 16
        DO 15 I=1,NUMGUB
        EPS(I)=0.
        IY=IYGUB(I)
        IYLAST=IY+IYELM(I)-1
        DO 15 II=IY,IYLAST
        EPS(I)=EPS(I)+Y(II)
15      CONTINUE
C       GUB SCALE FACTOR
        EPS(I)=(EPS(I)-1.)*10.
        IF(EPS(I).GT.TOL(3))IFEAS=0
        IF(EPS(I)*BETA(I).GT.TOL(3).OR.EPS(I)*BETA(I).LT.-
     *  TOL(3))ICOMP=0
        EPSI=EPSI+EPS(I)**2
10      CONTINUE
        IF(LP.GE.4)
       *PRINT*,"EPS=",(EPS(I),I=1,NUMGUB)
C-----COMPUTE STEP SIZE
16      CONTINUE
        IF(NUMPTH.EQ.0.AND.NUMGUB.EQ.0) GO TO 44
        IF(LP.GE.4)
       *PRINT*,"DEL IFEAS ICOMP ISTOP ICRITER ",
       * DEL,IFEAS,ICOMP,ISTOP,ICRITER
        IF(DEL+EPSI.LE.TOL(2)) GO TO 200
C-----STOPPING RULE
        IF(IFEAS*ICOMP.EQ.1) GO TO 200
        VGUESS=.5*(VINCUM+VLAGR)
C       DELTA=.00001
C       DELTB=.00001
C       IF(DEL.GE.TOL(2))DELTA=2.*(VGUESS-VLAST)/DEL
C       IF(EPSI.GE.TOL(2))DELTB=1.*(VGUESS-VLAST)/EPSI
C       SCALE FACTOR FOR STEP SIZE
        DELT=SCAL*(VGUESS-VLAST)/(DEL+EPSI)
        DELT=AMAX1(DELT,DMIN)
        DELTA=DELT
        DELTB=DELT
        IF(LP.GE.4)
       *PRINT*,"EPSI=",EPSI,"DELTA",DELTA,"DELTB",DELTB
C-----STOPPING RULE
        IF(DELTA+DELTB.LE..005) GO TO 200
C-----CHANGE ALPA(I) & BETA(I)
        IF(NUMPTH.EQ.0) GO TO 39
        DO 38 IW=1,NUMPTH
        NW=IPTBW(IW)
        IF(IW.LE.NUMBTH)IPSTS(NW)=2
        NOTDEL=1
        IF(IW.GT.NUMBTH) NOTDEL=0
        NUM=NUM1(NW)
```

```
      NUMEL=NUMELM(NW)+NUM-1
C-----DELETE FROM IPTH ANY NONBASIC W WITH ALL ALPA=0.
      DO 29 I=NUM,NUMEL
      NA=IPTH(I)
C     PATH CAPACITY SCALE FACTOR
      ALP=ALPA(I)+DELTA*(W(NW)-T(NW)*Y(NA))*PCSCAL/T(NW)
      IF(IYBSTS(NA).GE.4) ALP=0.
      ALPA(I)=AMAX1(0.,ALP)
C     PRINT*,"I NA ALP ALPA(I) ",I,NA,ALP,ALPA(I)
      IF(ALPA(I).LE.0.) GO TO 29
      NOTDEL=1
      IF(IW.LE.NUMBTH) IPSTS(NW)=3
29    CONTINUE
30    IF(NOTDEL.NE.1) CALL DELP(IW)
      IMAX=NUM1(NUMPTH)+NUMELM(NUMPTH)-1
      IF(LP.GE.4)
     *PRINT*,"NA ALPA",("/",IPTH(I),"/",ALPA(I),I=1,IMAX)
39    CONTINUE
C-----NO GUB'S CASE
      IF(NUMGUB.EQ.0) GO TO 46
      DO 40 I=1,NUMGUB
40    BETA(I)=AMAX1(0.,(BETA(I)+DELTB*EPS(I)))
      IF(LP.GE.4)
     *PRINT*,"BETA",("/",BETA(I),I=1,NUMGUB)
      GO TO 46
44    DELTA=1.0
      DELTB=1.0
C-----COMPUTE MODIFIED PATH COSTS
46    DO 45 J=1,NUMARC
45    FTIL(J)=TFCOST(J)
      IF(NUMPTH.EQ.0) GO TO 51
      DO 50 IW=1,NUMPTH
      NW=IPTBW(IW)
      NUM=NUM1(NW)
      NUMEL=NUMELM(NW)+NUM-1
      DO 50 I=NUM,NUMEL
      NA=IPTH(I)
      DTIL(NW)=VCOST(NA)+DTIL(NW)
      IF(ALPA(I).EQ.0) GO TO 50
      DTIL(NW)=DTIL(NW)+ALPA(I)*PCSCAL/T(NW)
C     PATH CAPACITY SCALE FACTOR
      FTIL(NA)=FTIL(NA)-ALPA(I)*PCSCAL
50    CONTINUE
51    CONTINUE
C----MODIFIED PATH COSTS WITH GUB
      BET=0.
      IF(NUMGUB.EQ.0) GO TO 53
C-----SAVE CONSTANT SUM OF ALL BETA(I)
      DO 65 J=1,NUMGUB
      IY=IYGUB(J)
```

```
      IYLAST=IY+IYELM(J)-1
      BET=BET+BETA(J)
      DO 65 II=IY,IYLAST
      FTIL(II)=FTIL(II)+BETA(J)*10.
65    CONTINUE
      BET=BET*10.
      IF(LP.GE.4)
     *PRINT*,"BET",BET
53    IF(NUMPTH.GT.0.AND.LP.GE.4)
     *PRINT *,"DTIL",("/",DTIL(J),
     *  J=1,NUMPTH)
      IF(LP.GE.4)
     *PRINT *,"FTIL",("/",FTIL(J),J=1,NUMARC)
C-----RECOMPUTE OBJ FOR NEW DTIL & FTIL
      OBJ=-BET+FFIX
      DO 61 J=1,NUMARC
      OBJ=OBJ+FTIL(J)*Y(J)
61    CONTINUE
      IF(NUMBTH.LE.0) GO TO 9
      DO 62 K=1,NUMBTH
      NW=IP1BW(K)
      OBJ=OBJ+DTIL(NW)*W(NW)
62    CONTINUE
C
      IF(LP.GE.2)
     *PRINT*,"NUMPTH NUMBTH ",NUMPTH,NUMBTH
C-----CHECK Y(J) CAN ENTER
9     CONTINUE
      ITLP=ITLP+1
      ITLPL=ITLPL+1
      IF(LP.GE.3)
     *PRINT*,"**********",
     *  "BEGIN LP ITER ",ITLP,"(NO ",ITLPL," OF LAGR)","OBJ=
     *  ",OBJ
      IF(LP.GE.4)
     *PRINT*,"NUMPTH NUMBTH IP1BW IPSTS ",NUMPTH,NUMBTH,
     * (IP1BW(IW),IPSTS(IW),IW=1,NUMPTH),"*********"
      CALL TABLX
      IF(LP.GE.4)
     *PRINT*,"IARTYP",("/",IARTYP(I),I=1,NUMARC)
      IF(LP.GE.4)
     *PRINT *,"IYBSTS",("/",IYBSTS(I),I=1,NUMARC)
      DO 70 J=1,NUMARC
      IF(IYBSTS(J).NE.1.AND.IYBSTS(J).NE.2) GO TO 70
      IF(IYBSTS(J).EQ.1.AND.IARTYP(J).EQ.-2) GO TO 71
      IF(IYBSTS(J).EQ.2.AND.IARTYP(J).EQ.-2) GO TO 73
C-----CASE OF IYBSTS(J)=1 OR 2 IN A ROW WITH W BASIC
      IBW=IARTYP(J)
      NW=IP1BW(IBW)
      ICOL=J
```

```
      CALL INVCOL(IBW)
      UMULT=-UBND(J)
      IF(IYBSTS(J).EQ.2)UMULT=-UMULT
      DO 74 I=1,NUMARC
74    COL(I)=COL(I)*UMULT
      IYENTER=1
      CALL REDUCED
      IF(IYBSTS(J).EQ.1)ZC=ZC-FTIL(J)
      IF(IYBSTS(J).EQ.2)ZC=ZC+FTIL(J)
      IF(ZC.LE.TOL(2)) GO TO 76
      IF(LP.GE.4)
     *PRINT*,"Y ENTER ZC J= ",J,ZC
      CALL MIN(ICOL,JSMAL)
      IF(JSMAL.EQ.0) GO TO 72
      CALL RHSIDE
      IF(IYBSTS(J).EQ.1) GO TO 407
      DO 408 I=1,NUMARC
408   COL(I)=-COL(I)
407   CALL PIVREV(ICOL,JSMAL)
72    IYENTER=0
      GO TO 9
71    ZC=-FTIL(J)
      IF(ZC.LE.TOL(2)) GO TO 70
C-----UPDATE BASIC STATUS OF Y(J),S(J) & OBJ
      IYBSTS(J)=2
      Y(J)=1.
      S(J)=UBND(J)
      RHS(J)=S(J)
      OBJ=OBJ+FTIL(J)
      IYENTER=0
      GO TO 9
C-----UPDATE BASIC STATUS OF Y(J),S(J) & OBJ
73    ZC=-FTIL(J)
      IF(ZC.GE.TOL(2)) GO TO 70
      IYBSTS(J)=3
      Y(J)=(UBND(J)-S(J))/UBND(J)
      S(J)=0.
      IARTYP(J)=-3
      RHS(J)=Y(J)
      OBJ=OBJ+(1.-Y(J))*(-FTIL(J))
      GO TO 9
76    IYENTER=0
70    CONTINUE
C
C-----CHECK S(J) CAN ENTER
      DO 60 J=1,NUMARC
      IF(IYBSTS(J).EQ.1.OR.IYBSTS(J).EQ.5) GO TO 60
      IF(IYBSTS(J).EQ.3) GO TO 61
      IBW=IARTYP(J)
      IF(IBW.LE.0) GO TO 60
```

```
      NW=IP1BW(IBW)
      ICOL=J
      CALL INVCOL(IBW)
      ISENTER=1
      CALL REDUCED
      IF(ZC.LE.TOL(2)) GO TO 86
      IF(LP.GE.4)
     *PRINT*,"S ENTER ZC J= ",J,ZC
      CALL MIN(ICOL,JSMAL)
      IF(JSMAL.EQ.0) GO TO 82
      CALL RHSIDE
      CALL PINVREV(ICOL,JSMAL)
82    ISENTER=0
      GO TO 9
81    ZC=-FTIL(J)/UBND(J)
      IF(ZC.LE.TOL(2)) GO TO 80
C-----UPDATE BASIC STATUS OF Y(J),S(J)& OBJ
C
      OLDY=Y(J)
      IYBSTS(J)=2
      Y(J)=1.
      S(J)=(1.-OLDY)*UBND(J)
      RHS(J)=S(J)
      IARTYP(J)=-2
      OBJ=OBJ+(1.-OLDY)*FTIL(J)
      GO TO 9
86    ISENTER=6
80    CONTINUE
C
C-----CHECK NONBASIC W IN IPTH ENTER
      IF(NUMBTH.EQ.NUMPTH.OR.NUMPTH.EQ.0) GO TO 90
      NN=NUMBTH+1
      DO 91 IW=NN,NUMPTH
      ICOL=IW
      NW=IP1BW(IW)
      CALL NPTHW(ICOL)
      IWENTER=1
      CALL REDUCED
      ZC=ZC-DTIL(NW)
      IF(ZC.LE.TOL(2))GO TO 94
      IF(LP.GE.4)
     *PRINT*,"IPTH W ENTER ZC IW NW ",ZC,IW,NW
      IPSTS(NW)=3
C     IPSTS(NW)=3
      CALL MIN(ICOL,JSMAL)
      IF(JSMAL.EQ.0) GO TO 92
C-----ADD A ROW & A COLUMN TO PINV
C      AND REVISE PINVERSE
      CALL RHSIDE
      CALL PINVREV(ICOL,JSMAL)
```

```
92      IWENTER=0
        GO TO 9
94      IWENTER=0
91      CONTINUE
C
C-----NEW PATH GENERATED
90      CONTINUE
C----MODIFIED ARC LENGTH
        CALL ARCLTH
C-----PREPARATION OF SHORT ROUTINE
        EAGLE=-1.
        CALL SHORT(EAGLE)
        ZC=-EAGLE
        IF(-EAGLE.LE.TOL(2)) GO TO 100
C-----COMPUTE & SAVE DTIL(NW),T(NW) &
C       ALPA(I)=0 FOR ALL I
        NW=NUMNEW
        W(NW)=0.
        T(NW)=INFIN
        DTIL(NW)=0.
        NUM=NUM1(NW)
        NUMEL=NUMELM(NW)+NUM-1
        DO 93 I=NUM,NUMEL
        NA=IPTH(I)
        DTIL(NW)=DTIL(NW)+VCOST(NA)
        IF(UBND(NA).LT.T(NW))T(NW)=UBND(NA)
93      ALPA(I)=0.
        ICOL=NUMPTH
        IPSTS(NW)=2
        IPTBW(NUMPTH)=NUMPTH
        CALL NPTHW(ICOL)
        IWENTER=1
        CALL MIN(ICOL,JSMAL)
        IF(JSMAL.EQ.0) GO TO 96
        CALL PHSIDE
        CALL PINVREV(ICOL,JSMAL)
96      IWENTER=0
        GO TO 9
C-----PRESENT SOLUTION IS OPTIMAL
100     VLAST=OBJ
        IF(ISFIR.NE.1) GO TO 102
        ISFIR=0
        VSTOP=VLAST
        GO TO 155
C-----STOPPING RULE
102     IF(LP.GE.2)
       *PRINT*,"VLAST VSTOP ",VLAST,VSTOP
        IF((VLAST-VSTOP)/VSTOP.LT.TOL(4)) GO TO 150
        VSTOP=VLAST
        ISTOP=0
```

```
      GO TO 155
C-----CONVERT ALL SOLUTIONS TO NODE-ARC FORMS
150   IF(VLAST.LE.VLAGR) GO TO 198
155   VLAGR=VLAST
      IF(LP.GE.2)
     *PRINT*,"VLAGR ",VLAGR
      CALL XSOL
      CALL FNDSOL(VRND)
      CALL BBFEAS(VRND,X,XINCUM,IRTNST)
      IF(VINCUM.GT.VRND)VINCUM=VRND
      GO TO (210,301,800),IRTNST
C-----CHECK FOR FATHOMING VIA BOUND
210   CALL BBCKBD(VLAGR,IRTNST)
      GO TO (198,301,800),IRTNST
C-----CHECK STOPPING RULE
198   ISTOP=ISTOP+1
      IF(ISTOP.GE.ICRITER) GO TO 200
      GO TO 1
C
200   CONTINUE
C-----SELECT PARTIONING VARIABLE
C     FIRST FREE & FRACTION VARIABLE
300   CONTINUE
      JPARTN=0
      DO 280 J=1,NUMARC
      IF(FCOST(J).LE.TOL(5)) GO TO 280
      IF(IFIXST(J).NE.0) GO TO 280
      IF(YLAGR(J).LE.TOL(3).OR.YLAGR(J).GE.1.-TOL(3)) GO TO
     *  370
C-----Y IS FREE & FRAC.
      JPARTN=J
      GO TO 290
C-----Y IS FREE & INTEGER
370   JPARTN=J
280   CONTINUE
      IF(JPARTN.NE.0) GO TO 290
C-----ALL VARIABLES FIXED
      CALL BBCKBD(XINFIN,IRTNST)
      GO TO (500,301,800),IRTNST
290   CALL BBPRTN(JPARTN,.5,VLAGR,DUMMY,VLAGR,DUMMY,
     *IFIXST,REF,REF,IRTNST)
      GO TO (301,301,301,301,800),IRTNST
301   CALL BBSELC(END,AUX,IFIXST,REF,REF,IRTNST)
      FIRTIM=SECOND(P)
      IF(FIRTIM.LT.TINFIN) GO TO 302
      GO TO 303
302   TINFIN=FIRTIM
      FIRST=FIRTIM
      FIRTIME=INT((FIRTIM-APFIN)*1000+.5)/1000.
      IFIRLP=ITLP-ITLPAP
```

```
303     ISTOP=0                                                          99
        ISFIX=1
C       REDUCE ICITER BY ONE HALF AFTER SECOND SELECTION
        IF(INCRIT.EQ.1)ICRITER=ICRITER/2+1
        INCRIT=0
        FFIX=0.
        VLAGR=AMIN1(VLAGR,VINCOM)
C-----REVISE TFCOST & FFIX
        DO 323 J=1,NUMARC
323     TFCOST(J)=FCOST(J)
        DO 310 J=1,NUMARC
        IF(FCOST(J).LE.TOL(5)) GO TO 310
        IF(IFIXST(J).EQ.0) GO TO 310
        IF(IFIXST(J).EQ.1) GO TO 315
        TFCOST(J)=0.
        FFIX=FFIX+FCOST(J)
C-----CLOSE ANY OTHER MEMBER OF A GUB SET
        DO 308 I=1,NUMGUB
        IF(J.GE.IYGUB(I).AND.J.LT.IYGUB(I)+IYELM(I)) GO TO
308     CONTINUE
        GO TO 310
309     IY=IYGUB(I)
        IYEND=IY+IYELM(I)-1
        DO 307 I=IY,IYEND
        IF(I.EQ.J) GO TO 307
        TFCOST(I)=XINFIN
307     CONTINUE
        GO TO 310
315     TFCOST(J)=INFIN
310     CONTINUE
        IF(LP.GE.2)
       *PRINT *,"TFCOST",("/",TFCOST(J),J=1,NUMARC)
        GO TO 1
800     CALL BBEND(OPSOLV,TIMTOT,NUMSEL,IRTNST)
        GO TO (850,880),IRTNST
C
C-----WRITE ALL SOLUTIONS-----
C
850     FINIS=SECOND(P)
        TIME=INT((FINIS-FIRST)*1000+.5)/1000.
        TOTAL=INT((FINIS-APSTAT)*1000+.5)/1000.
        IFINAL=ITLP-IFIRLP
        WRITE(6,810)NAMPRB,NUMNODE,NUMARC,NUMFCA,NUMGUB
        WRITE(6,540)ICRITER,LP,SCAL,PCSCAL,DMIN
        PRINT*,"NO. OF ITERATIONS IN APBASIS=",ITLPAP
        PRINT*,"TIME FOR APBASIS=",APTIME
        PRINT*,"NO. OF LP ITERATIONS UP TO FIRST=",IFIRLP
        PRINT*,"TIME FROM APBASIS TO FIRST=",FIRTIME
        PRINT*,"NO. OF LP ITERATIONS AFTER SELECTION=",IFINAL
        TWORK=INT((FINIS-FIRST)*1000.+.5)/1000.
```

```
      PRINT*,"TIME FROM SELECTION TO LAST= ",TWORK
      PRINT*,"TOTAL TIME FROM AP TO LAST=",FINIS-APSTAT
      PRINT*,"OPTIMAL SOLUTION=",OPSOLV                                  100
      WRITE(6,815)
      WRITE(6,820)(I,IBEG(I),IEND(I),LBND(I),UBND(I),
     *  VCOST(I),
     *FCOST(I),XX(I),XINCUM(I),I=1,NUMARC)
      IF(NUMGUB.EQ.0) GO TO 955
      WRITE(6,825)
      WRITE(6,830)(I,IYGUB(I),IYELM(I),I=1,NUMGUB)
      GO TO 955
880   WRITE(6,3105)
3105  FORMAT("0*---INFEASIBLE SOLUTION---")
955   STOP
      END
```

```
C
      SUBROUTINE APBASIS
C
C-----SUBROUTINE CONSTRUCT ARC-PATH BASIS FROM NODE-ARC
C     SOLUTIONS
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *  ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAP(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),FCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *  ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),IPS(100),
     *  OBJ,ITLP,
     *Q(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,150)
      INTEGER UBND
      NUMBTH=0
      OBJ=0.
      NUMPTH=0
      VLAST=0.
```

```
C-----ASSIGN TEMPORARY UPPER BOUND AND ARC-COSTS BASED
C     ON NODE-ARC SOLUTIONS
      DO 10 I=1,NUMARC
      IF(X(I).GE.TOL(3)) GO TO 11
      TVCOST(I)=INFIN
      Y(I)=0.
      IYBSTS(I)=1
      IF(FCOST(I).LE.TOL(3))GO TO 25
      S(I)=0.
      RHS(I)=0.
      IARTYP(I)=-2
C     IF(FCOST(I).GT.TOL(3))IYBSTS(I)=3
C     IF(FCOST(I).GT.TOL(3))IARTYP(I)=-3
CC    PRINT*,"X IARTYP TVCOST RHS ",X(I),IARTYP(I),TVCOST(I)
C       ,RHS(I)
      GO TO 10
11    TVCOST(I)=VCOST(I)+FCOST(I)/UBND(I)
      Y(I)=X(I)/UBND(I)
      IF(FCOST(I).LE.TOL(3)) Y(I)=0.
      VLAST=VLAST+X(I)*VCOST(I)+Y(I)*FCOST(I)
      IF(Y(I).GE.TOL(3).AND.FCOST(I).GE.TOL(3)) GO TO 12
25    IARTYP(I)=-2
      S(I)=UBND(I)
      IYBSTS(I)=5
      IF(UBND(I).NE.LBND(I)) IYBSTS(I)=4
      RHS(I)=S(I)
CC    PRINT*,"X IARTYP TVCOST RHS ",X(I),IARTYP(I),TVCOST(I)
C       ,RHS(I)
      GO TO 10
12    IARTYP(I)=-3
      Y(I)=0.
      S(I)=0.
      RHS(I)=Y(I)
      IYBSTS(I)=3
CC    PRINT*,"X IARTYP TVCOST RHS ",X(I),IARTYP(I),TVCOST(I)
C       ,RHS(I)
10    CONTINUE
C-----CREATE A SHORTEST PATH
14    CONTINUE
      ITLP=ITLP+1
      CALL TABLX
      EAGLE=1.
      CALL SHORT(EAGLE)
      ZO=EAGLE
C-----CHECK ALL X SOLUTIONS USED UP OR NOT
      IF(EAGLE.GE.INFIN) GO TO 901
C-----COMPUTE & SAVE DTIL(NW) , T(NW) & A_PA(I)=0.
C     FOR ALL I
      NW=NUMNEW
      UMAX=INFIN
```

```
      XMAX=INFIN
      T(NW)=INFIN
      W(NW)=0.
      OTIL(NW)=0.
      NUM=NUM1(NW)
      NUMEL=NUMELM(NW)+NUM-1
      DO 15 I=NUM,NUMEL
      NA=IPTH(I)
      OTIL(NW)=OTIL(NW)+VCOST(NA)
      IF(UBND(NA).LT.T(NW))T(NW)=UBND(NA)
      XMAX=AMIN1(XMAX,X(NA))
      IF(IYBSTS(NA).GE.4) UMAX=AMIN1(UMAX,S(NA))
      IF(IYBSTS(NA).LT.4) UMAX=AMIN1(UMAX,(1.-Y(NA))
     *  *UBND(NA))
15    ALPA(I)=0.
      IF(LP.GE.2)
     *PRINT*,"XMAX UMAX ",XMAX,UMAX
      ICOL=NUMPTH
      IPSTS(NW)=2
      IP1BW(NUMPTH)=NUMPTH
      IF(XMAX.GE.UMAX) GO TO 30
C-----FORBID W NOT REACHING SOME CAPACITY
      IPSTS(NW)=1
      GO TO 14
C-----CREATE A NONBASIC COLUMN
30    CALL NPTHW(ICOL)
      IWENTER=1
      CALL MIN(ICOL,JSMAL)
      CALL RHSIDE
      CALL PINVREV(ICOL,JSMAL)
      IWENTER=0
C-----RESET TEMPORARY UPPER BOUND AND COSTS
      DO 20 I=NUM,NUMEL
      NA=IPTH(I)
      X(NA)=X(NA)-W(NW)
      IF(X(NA).LE.TOL(3)) TVCOST(NA)=INFIN
20    CONTINUE
      DO 35 ID=1,NUMPTH
      IW=IP1BW(ID)
CC    PRINT*,"NUMPTH IW IPSTS(IW) ",NUMPTH,IW,IPSTS(IW)
      IF(IPSTS(IW).EQ.1) CALL DELP(ID)
35    CONTINUE
      GO TO 14
900   DO 40 I=1,NUMARC
      IF(IYBSTS(I).NE.5) GO TO 40
      IF(IARTYP(I).NE.-2) GO TO 40
      PRINT*,"ARTIFICIAL IN SOLUTION"
      STOP 1
40    CONTINUE
      RETURN
```

```
      END
```



```
C
      SUBROUTINE ARCLTH
C
C-----SUBROUTINE HANDLE TO MODIFY ARC LENGTH
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *   IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *   SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *   OBJ,ITLP,
     *QL(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *   SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,150)
      INTEGER UBND
      DO 10 NA=1,NUMARC
      IF(IARTYP(NA).GT.0) GO TO 20
      IF(IARTYP(NA).EQ.-2) GO TO 25
      IF(IARTYP(NA).EQ.-3) GO TO 30
20    IW=IARTYP(NA)
      NW=IP1BW(IW)
      CALL INVCOL(IW)
      TVCOST(NA)=VCOST(NA)
      DO 22 I=1,NUMARC
      JW=IARTYP(I)
      IF(JW.EQ.-2) GO TO 22
      IF(JW.EQ.-3) GO TO 23
      MW=IP1BW(JW)
      TVCOST(NA)=TVCOST(NA)-DTIL(MW)*COL(I)
      GO TO 22
23    TVCOST(NA)=TVCOST(NA)-FTIL(I)*COL(I)
22    CONTINUE
      GO TO 10
30    TVCOST(NA)=VCOST(NA)+FTIL(NA)/UBND(NA)
```

```
      GO TO 10
25    TVCOST(NA)=VCOST(NA)
10    CONTINUE
C     PRINT *,"TVCOST",("/",TVCOST(I),I=1,NUMARC)
      RETURN
      END




C
      SUBROUTINE DELP(ID)
C
C-----SUBROUTINE DELETE NWR FROM IPTH LIST
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *  ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *  ,
     *FTIL(500),IYGUB(100),IYLLM(100),BETA(100),EPS(100),
     *  OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(500),NUMGJB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,150)
      INTEGER UBND
      NWR=IP1BW(ID)
      IF(LP.GE.4)
     *PRINT*,"DELETE ID NWR ",ID,NWR
      DO 10 NW=1,NUMPTH
      IF(IP1BW(NW).GT.NWR) IP1BW(NW)=IP1BW(NW)-1
      IF(NW.LE.ID) GO TO 10
C-----SLIDE IPTH LIST FOR NW GREATER THAN ID
C     IARTYP,IP1BW & KARTYP
      IP1BW(NW-1)=IP1BW(NW)
      KARTYP(NW-1)=KARTYP(NW)
      NA=KARTYP(NW-1)
      IF(LP.GE.4)
     *PRINT*,"NA IARTYP(NA) ",NA,IARTYP(NA)
      IF(NW.LE.NUMBTH)IARTYP(NA)=IARTYP(NA)-1
```

```
      IF(LP.GE.4)
     *PRINT*,"IP1BW(NW-1) KARTYP(NW-1) IARTYP(NA) ",
     *  IP1BW(NW-1),                                                    105
     * KARTYP(NW-1),IARTYP(NA)
10    CONTINUE
      IF(NWR.EQ.NUMPTH) GO TO 900
      NWBEG=NWR+1
      IOFFST=NUMELM(NWR)
      DO 100 NW=NWBEG,NUMPTH
      NUM1(NW-1)=NUM1(NW)
      NUMELM(NW-1)=NUMELM(NW)
      W(NW-1)=W(NW)
      DTIL(NW-1)=DTIL(NW)
      T(NW-1)=T(NW)
      IPSTS(NW-1)=IPSTS(NW)
      IBEGG=NUM1(NW-1)
      IENDD=IBEGG+NUMELM(NW-1)-1
      NUM1(NW-1)=NUM1(NW-1)-IOFFST
      DO 100 I=IBEGG,IENDD
      IPTH(I-IOFFST)=IPTH(I)
      ALPA(I-IOFFST)=ALPA(I)
100   CONTINUE
900   NUMPTH=NUMPTH-1
      IF(ID.LE.NUMBTH) NUMBTH=NUMBTH-1
      IF(LP.GE.4)
     *PRINT*,"AFTER NUMBTH=",NUMBTH
      RETURN
      END




C
      SUBROUTINE INVCOL(ICOL)
C-----SUBROUTINE GENERATE ONE COLUMN OF PINV
C
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *  ,
     *IBEG(500),IEND(500),IPTH(50000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(50..),
```

```
      *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
      *   ,
      *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
      *  JBJ,IT_P,
      *Q_(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
      *  SMAL,ZC,LP
       COMMON BB(60),BBTAB(4,1500)
       INTEGER UBND
C-----INITIALIZE
       DO 10 I=1,NUMARC
10     COL(I)=0.
       JJ=ICOL
C-----PICK A COLUMN OF PINV
       DO 20 IW=1,NUMBTH
       NW=IPIBW(IW)
       NUM=NUM1(NW)
       NUMEL=NUMELM(NW)+NUM-1
       PIN=PINV(IW,JJ)
       IF(PIN.EQ.0.) GO TO 20
       DO 36 I=NUM,NUMEL
       NA=IPTH(I)
       IF(IARTYP(NA).EQ.-2) GO TO 32
       IF(IARTYP(NA).EQ.-3) GO TO 33
C----P1 TYPE
       GO TO 36
C----P2 TYPE
32     COL(NA)=COL(NA)-PIN
       GO TO 36
C----P3 TYPE
33     COL(NA)=COL(NA)+PIN/UBND(NA)
36     CONTINUE
C-----COPY DIAGONAL ELEMENT OF PINV
       NA=KARTYP(IW)
       COL(NA)=PIN
20     CONTINUE
C      PRINT*,"INVCOL",("/",COL(I),I=1,NUMARC)
       RETURN
       END




C
       SUBROUTINE MIN(ICOL,JSMAL)
C-----SUBROUTINE DETERMINE LEAVING ROW & MIN RATIO
       COMMON/AAPDAT/X(500),XINCUM(500),X_AGR(500),YLAGR(500)
      *   ,
      *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
      *  FCOST(500),
```

```
      *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
      *  IWENTER,
      *NUMPTH,NUMOAR(150),NUM1(150),IFIRARC(150),VLAST,
      *  NUMNEW,
      *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
      *  IARBP,
      *IARTYP(500),COL(500),RHS(500),IPIBW(100),S(500),INFIN,
      *  SCOL(500),
      *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
      *  ,
      *FTIL(500),IYGUB(100),IYELM(200),BETA(100),EPS(100),
      *  OBJ,ITLP,
      *Q1(75),PINV(75,75),ALPA(5000),NUMGJB,NUMBTH,TOL(5),
      *  SMAL,ZC,LP
       COMMON BB(60),BBTAB(4,150)
       INTEGER UBND
       SMAL=INFIN
       ISSLK=0
       JSMAL=0
       DO 10 J=1,NUMARC
       JMULT=1
       IF(IARTYP(J).GT.0) IBW=IARTYP(J)
       IF(COL(J).LE.-TOL(1)) GO TO 70
       IF(COL(J).GE.TOL(1)) GO TO 15
       GO TO 10
70     IF(IARTYP(J).NE.-3) GO TO 10
       RAT=(1.-RHS(J))/(-COL(J))
       JMULT=-1
       GO TO 20
15     RAT=RHS(J)/COL(J)
20     IF(RAT-SMAL) 30,40,10
30     IF(IARTYP(J).EQ.-2.AND.IYBSTS(J).NE.5) ISSLK=1
       JSMAL=J*JMULT
       SMAL=RAT
       GO TO 10
40     IF(ISSLK.EQ.0.OR.IARTYP(J).EQ.-2) GO TO 10
       ISSLK=0
       JSMAL=J*JMULT
       SMAL=RAT
       GO TO 10
10     CONTINUE
C-----CHECK MIN EQUAL 0 BETWEEN Y(J) AND S(J)
C      OR MIN=0 AND JSMAL=0
       IF(JSMAL.EQ.0) GO TO 11
       IF(IYENTER.EQ.1.AND.SMAL.GE.1.-TOL(3)) GO TO 11
       GO TO 9
11     IF(IYBSTS(ICOL).EQ.1) GO TO 17
       Y(ICOL)=0.
       IYBSTS(ICOL)=1
       XMULT=UBND(ICOL)
```

```
      IF(IARTYP(ICOL).NE.-2) GO TO 50
      S(ICOL)=0.
      RHS(ICOL)=S(ICOL)
      OBJ=OBJ-ZC
      JSMAL=0
      GO TO 9
17    Y(ICOL)=1.
      IYBSTS(ICOL)=2
      XMULT=-UBND(ICOL)
      IF(IARTYP(ICOL).NE.-2) GO TO 50
      S(ICOL)=UBND(ICOL)
      RHS(ICOL)=S(ICOL)
      OBJ=OBJ-ZC
      JSMAL=0
      GO TO 9
C-----Y BLOCKS ITSELF IN NON-TRIVIAL PIVOT
50    ID=IARTYP(ICOL)
      CALL INVCOL(ID)
      DO 52 J=1,NUMARC
      RHS(J)=RHS(J)-XMULT*COL(J)
      ID=IARTYP(J)
      IF(ID.EQ.-2)S(J)=RHS(J)
      IF(ID.EQ.-3)Y(J)=RHS(J)
      NW=IPIBW(ID)
      IF(ID.GT.0)W(NW)=RHS(J)
52    CONTINUE
      OBJ=OBJ-ZC
      JSMAL=0
9     IF(LP.GE.4)
     *PRINT*,"JSMAL SMAL ",JSMAL,SMAL
      RETURN
      END
C
      SUBROUTINE NPTHW(ICOL)
C
C-----SUBROUTINE HANDLE EITHER NONBASIC W OR NEW PATH
C     COLUMN GENERATION
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(50000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
```

```
      *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
      *  IARBP,
      *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
      *  SCOL(500),
      *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
      *  ,
      *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
      *  OBJ,ITLP,
      *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
      *  SMAL,ZC,LP
       COMMON BB(50),BBTAB(4,1500)
       INTEGER UBND
C-----INITIALIZE
       NW=IP1BW(ICOL)
       DO 5 I=1,NUMARC
5      TCOL(I)=0.
       K=0
       NUM=NUM1(NW)
       NUMEL=NUMELM(NW)+NUM-1
C-----SCAN IPTH TO ASSIGN LEFT SIDE OF COLUMN
       DO 10 I=NUM,NUMEL
       NA=IPTH(I)
       IF(IARTYP(NA).GT.0) GO TO 11
       IF(IARTYP(NA).EQ.-2) GO TO 12
       IF(IARTYP(NA).EQ.-3) GO TO 13
C-----P1 TYPE & LIST OF Q1
11     Q1(K+1)=NA
       K=K+1
       GO TO 10
C-----P2 TYPE
12     TCOL(NA)=1.
       GO TO 10
C-----P3 TYPE
13     TCOL(NA)=-1./UBND(NA)
10     CONTINUE
C-----CALL INVCOL ROUTINE FOR EACH NON ZERO ENTRY IN Q1
       IF(K.EQ.0) GO TO 45
       DO 20 II=1,K
       NA=Q1(II)
       IICOL=IARTYP(NA)
       CALL INVCOL(IICOL)
       DO 30 I=1,NUMARC
30     TCOL(I)=TCOL(I)+COL(I)
20     CONTINUE
45     DO 50 I=1,NUMARC
       COL(I)=TCOL(I)
50     CONTINUE
C      PRINT *,"NPTH COL",("/",COL(I),I=1,NUMARC)
       RETURN
       END
```

```
C
      SUBROUTINE PINVREV(ICOL,JSMAL)
C                                                                      110
C-----SUBROUTINE REVISES PINVERSE DEPENDING ON LEAVING ROW
C
      COMMON/AAPOAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *   IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *   SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *   OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *   SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,1500)
      INTEGER UBND
      NR=IABS(JSMAL)
      IB1=NUMBTH+1
      IF(NUMBTH.EQ.0) GO TO 90
C-----PIVOT ELEMENT
      PELE=COL(NR)
C-----SAVE PART OF COL OPPOSITE P1
      DO 5 NA=1,NUMARC
      IF(IARTYP(NA).GT.0) GO TO 6
      GO TO 5
6     IW=IARTYP(NA)
      TCOL(IW)=COL(NA)
      IF(NA.EQ.NR) NRK=IW
CC    PRINT*,"NRK=IW  ",NA,IW,IARTYP(NA)
5     CONTINUE
C     IF(NUMBTH.GT.0)PRINT *,"PELE,TCOL",PELE,("/",TCOL(IW),
C    *   IW=1,NUMBTH)
      IF(LP.GE.4)
     *PRINT*,"UPDATE COL",("/",COL(NA),NA=1,NUMARC)
C-----SAVE SUB. COL IF REQUIRED
      IF(IWENTER.EQ.1.OR.NR.EQ.ICOL) GO TO 39
      IF(IARTYP(NR).LE.0) GO TO 50
      IW=IARTYP(NR)
      CALL INVCOL(IW)
      SPELE=COL(NR)
      DO 35 NA=1,NUMARC
```

```
      IF(IARTYP(NA).GT.0)GO TO 34
      GO TO 35
34    IW=IARTYP(NA)
      SCOL(IW)=COL(NA)
35    CONTINUE
      GO TO 39
50    IF(IARTYP(NR).EQ.-2)SPELE=1.
      IF(IARTYP(NR).EQ.-3)SPELE=-1./JBND(NR)
      DO 52 IW=1,NUMBTH
52    SCOL(IW)=0.
C-----CHECK NR A MEMBER OF PINV OR NOT
39    IF(IARTYP(NR).GT.0) GO TO 20
      DO 15 J=1,NUMBTH
      CALL INVCOL(J)
      DO 10 I=1,NUMBTH
      IF(ABS(TCOL(I)).LE.TOL(1)) GO TO 10
      PINV(I,J)=PINV(I,J)-COL(NR)/PELE*TCOL(I)
10    CONTINUE
      IF(IWENTER.EQ.0) GO TO 15
      PINV(IB1,J)=COL(NR)/PELE
15    CONTINUE
      IF(IWENTER.EQ.1) GO TO 38
      IF(NR.EQ.ICOL)GO TO 90
      DO 37 I=1,NUMBTH
      IF(ABS(TCOL(I)).LE.TOL(1)) GO TO 37
      SCOL(I)=SCOL(I)-SPELE/PELE*TCOL(I)
37    CONTINUE
      GO TO 90
38    IF(IARTYP(NR).EQ.-2) TCOL(IB1)=1.0
      IF(IARTYP(NR).EQ.-3) TCOL(IB1)=-1./JBND(NR)
      DO 17 I=1,NUMBTH
17    PINV(I,IB1)=-TCOL(IB1)/PELE*TCOL(I)
      PINV(IB1,IB1)=TCOL(IB1)/PELE
      GO TO 90
C-----NR AS A MEMBER OF PINV
20    CONTINUE
      DO 25 J=1,NUMBTH
      TEMP=PINV(NRK,J)
      DO 25 I=1,NUMBTH
      IF(ABS(TCOL(I)).LE.TOL(1)) GO TO 25
      IF(I.EQ.NRK) GO TO 24
      PINV(I,J)=PINV(I,J)-TEMP/PELE*TCOL(I)
      GO TO 25
24    PINV(I,J)=PINV(I,J)/PELE
      IF(IWENTER.EQ.1) PINV(IB1,J)=PINV(I,J)
25    CONTINUE
      IF(IWENTER.EQ.1) GO TO 42
      IF(NR.EQ.ICOL) GO TO 90
      DO 43 I=1,NUMBTH
      IF(ABS(TCOL(I)).LE.TOL(1))GO TO 43
```

```
      IF(I.EQ.NRK)GO TO 44
      SCOL(I)=SCOL(I)-SPELE/PELE*TCOL(I)
      GO TO 43
44    SCOL(I)=SCOL(I)/PELE
43    CONTINUE
      GO TO 90
42    IF(IARTYP(NR).LE.0) GO TO 90
      DO 40 I=1,IB1
40    PINV(I,IB1)=PINV(I,NRK)
90    IF(NUMBTH.EQ.0)PINV(IB1,IB1)=1.
C     DO 91 I=1,IB1
C     PRINT *,"PINV MATRIX",("/",PINV(I,J),J=1,IB1)
C91    CONTINUE
      CALL UPDATE(ICOL,JSMAL)
      RETURN
      END




C
      SUBROUTINE REDUCED
C-----SUBROUTINE COMPUTE REDUCED COSTS
C
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *   IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *   SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYLLM(100),BETA(100),EPS(100),
     *   OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGJB,NUMBTH,TOL(5),
     *   SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,150)
      INTEGER UBND
      ZC=0.
      DO 10 NA=1,NUMARC
      IF(IARTYP(NA).GT.0) GO TO 5
      IF(IARTYP(NA).EQ.-2) GO TO 10
```

```
      IF(IARTYP(NA).EQ.-3) GO TO 6
C-----PICK APPROPRIATE NW FOR NA
5     IBW=IARTYP(NA)                                                    113
      NW=IP1BW(IBW)
      ZC=ZC+DTIL(NW)*COL(NA)
      GO TO 10
6     ZC=ZC+FTIL(NA)*COL(NA)
10    CONTINUE
C     PRINT*,"REDUCED ZC ",ZC
      RETURN
      END




C
      SUBROUTINE RHSIDE
C
C-----SUBROUTINE UPDATE RHSIDE COLUMNS
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *  OBJ,ITER,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,1500)
      INTEGER UBND
      DO 10 I=1,NUMARC
      IF(IARTYP(I).GT.0) GO TO 5
      IF(IARTYP(I).EQ.-2) GO TO 6
      IF(IARTYP(I).EQ.-3) GO TO 7
5     IBW=IARTYP(I)
      NW=IP1BW(IBW)
      RHS(I)=RHS(I)-COL(I)*SMAL
      W(NW)=RHS(I)
      GO TO 10
```

```
6       RHS(I)=RHS(I)-COL(I)*SMAL
        S(I)=RHS(I)
        GO TO 10
7       RHS(I)=RHS(I)-COL(I)*SMAL
        Y(I)=RHS(I)
10      CONTINUE
        RETURN
        END




C
        SUBROUTINE RNDSOL(VRND)
C
        COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
       *   ,
       *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
       *   FCOST(530),
       *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
       *   IWENTER,
       *NJMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
       *   NUMNEW,
       *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
       *   IARBP,
       *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
       *   SCOL(500),
       *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
       *   ,
       *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
       *   OBJ,ITER,
       *QL(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
       *   SMAL,ZC,LP
        COMMON BB(60),BBTAB(4,150)
        INTEGER UBND
        DIMENSION YR(500)
C
C-----SECTION TO OBTAIN ROUND SOLUTION-----
C
        DO 210 J=1,NUMARC
        X(J)=XLAGR(J)
        YR(J)=0.
        IF(X(J).GE.TOL(3)) YR(J)=1.
210     CONTINUE
C       GUB CONSTRAINTS
        IF(NUMGUB.EQ.0) GO TO 225
        DO 215 J=1,NUMGUB
        IY=IYGUB(J)
```

```
      IYLAST=IY+IYELM(J)-1
      XX=0.
      SMAL=INFIN
      DO 216 I=IY,IYLAST
216   XX=XX+X(I)
      DO 217 II=IY,IYLAST
      TC=FCOST(II)+VCOST(II)*XX
      IF(TC-SMAL) 218,217,217
218   SMAL=TC
      JSMAL=II
217   CONTINUE
      DO 220 III=IY,IYLAST
      IF(III.EQ.JSMAL) GO TO 221
222   X(III)=0.
      YR(III)=0.
      GO TO 220
221   IF(XX.LE.TOL(3)) GO TO 222
      X(III)=XX
      YR(III)=1.
220   CONTINUE
215   CONTINUE
C-----OBTAIN ROUND VALUE
225   VRND=0.
      DO 240 I=1,NUMARC
240   VRND=VRND+VCOST(I)*X(I)+FCOST(I)*YR(I)
      RETURN
      END
```

```
C
      SUBROUTINE SHORT(EAGLE1)
C-----TO SOLVE SHORTEST FORBIDDEN PATH PROBLEMS BY A
C     MODIFIED DIJKSTRA'S PROCEDURE
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *   IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *   SCOL(500),
     *IPSTS(1300 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
```

```
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *  OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGJB,NUMBTH,TOL(5),                116
     *  SMAL,ZC,LP
      COMMON BB(600),BBTAB(4,1500)
C
      DIMENSION BARDIS(100),LSTPTH(500),KARC(500),
     *EAGLE(150),TB(150),TARC(500),FBPTH(150),TFBPTH(500),
     *TEAGLE(500),IGROW(500),NPOUT(100),NXTARC(100),
     *TTBPTH(500),TBARC(150),TIPTH(300)
      INTEGER TIPTH,UBND,TB,TARC,FBPTH,TFBPTH,TTBPTH,TBARC
C
C     PRINT*,NUMARC,(IBEG(NA),IEND(NA),LBND(NA),UBND(NA),
C    *VCOST(NA),FCOST(NA),NA=1,NUMARC)
C-----READ FORBIDDEN PATH INFORMATIONS
C     PRINT*,NUMPTH
C     LASTP=0
      START=SECOND(P)
C     DO 100 NP=1,NUMPTH
C-----READ NUMBER OF PATH ELEMENTS
C     PRINT*, NUMELM(NP)
C     IFIRP=LASTP+1
C     NUM1(NP)=IFIRP
C100  LASTP=IFIRP+NUMELM(NP)-1
C100  PRINT*,(IPTH(I),I=IFIRP,LASTP)
C-----NUMBER OF NODES
C-----SET UP
C     PRINT*,(I,IFIRARC(I),NUMOAR(I),I=1,NUMNODE)
      LST=0
      IF(NUMPTH.EQ.0) GO TO 151
      DO 150 J=1,NUMPTH
      IF(IPSTS(J).EQ.2) GO TO 150
      NXTARC(J)=NUM1(J)
      IF(LST.NE.0) GO TO 145
      LST=J
      IFIR=J
      GO TO 150
145   NPOUT(LST)=J
      LST=J
150   CONTINUE
      NPOUT(LST)=0
C     NXTARC(J)=NUM1(J)
C150  NPOUT(J)=J+1
C     NPOUT(NUMPTH)=0
 151  DO 200 I=1,NUMNODE
      FBPTH(I)=0
 200  EAGLE(I)=INFIN
      BARDIS(IFIR)=0
      IF(LST.EQ.0) EAGLE(1)=0.
      FBPTH(1)=IFIR
```

```
      IF(LST.EQ.0)  FBPTH(1)=0
C-----MAIN LOOP BEGINS
C
      ND=1
C
   99 IF(ND.EQ.NUMNODE) GO TO 899
C     PRINT*,"NUMPTH NPOUT BARDIS ",NUMPTH,(NPOUT(NX),
C       BARDIS(NX),
C    * NX=1,NUMPTH)
C     PRINT*,"ND EAGLE FBPTH ",ND,(EAGLE(NX),FBPTH(NX),
C    * NX=1,NUMNODE)
C-----ASSIGN IGROW,TEAGLE & TFBPTH TO ALL ARCS REACHABLE
      J=FBPTH(ND)
      MAXK=NUMOAR(ND)
      NA=IFIRARC(ND)
      IF(J.NE.0) GO TO 1100
C-----EMPTY FLAG CASE
      DO 1000 K=1,MAXK
      TARC(K)=NA
      TEAGLE(K)=EAGLE(ND)
      TFBPTH(K)=0
      TTBPTH(K)=0
      NA=NA+1
 1000 CONTINUE
      GO TO 550
 1100 DO 190 K=1,MAXK
      LSTPTH(K)=0
      KARC(NA)=K
      TARC(K)=NA
      IGROW(K)=1
      TEAGLE(K)=BARDIS(J)
      TFBPTH(K)=0
      TTBPTH(K)=J
      NA=NA+1
  190 CONTINUE
C     PRINT*,(KK,TARC(KK),IGROW(KK),TEAGLE(KK),TFBPTH(KK),
C    *KK=1,MAXK)
C
      ITER=0
C
C-----SCAN FOR NEXT BAR DISTANCE
      IGREW=1
  201 IF(IGREW.EQ.0) GO TO 550
      IGREW=0
      JSAV=J
      ITER=ITER+1
  205 IF(JSAV)210,220,230
  210 JSAV=-JSAV
      BARSV=BARDIS(JSAV)
      GO TO 240
```

```
 220   BARSV=EAGLE(ND)
       GO TO 240
 230   JSAV=NPOUT(JSAV)
C-----SCAN BAR
       GO TO 205
 240   NXT=NPOUT(J)
       L=NXTARC(J)
       NA=IPTH(L)
       K=KARC(NA)
       JJ=LSTPTH(K)
C-----FLAG SUBDIVISION TO APPROPRIATE ARCS
       IF(IGROW(K)-ITER)500,400,300
C      FIRST FLAG SUBDIVISION
 400   BARDIS(J)=TEAGLE(K)
       TEAGLE(K)=BARSV
       IF(TFBPTH(K).EQ.0) TFBPTH(K)=J
       TTBPTH(K)=JSAV
       IF(JJ.NE.0)NPOUT(JJ)=-J
       IGROW(K)=IGROW(K)+1
       IGREW=1
       GO TO 450
 300   NPOUT(JJ)=J
 450   NPOUT(J)=0
C-----MOVE TO NEXT ARC POSITION
       NXTARC(J)=NXTARC(J)+1
       LSTPTH(K)=J
 500   J=NXT
C      PRINT*,FBPTH(ND),JSAV,K,TEAGLE(K),TFBPTH(K),
C     *LSTPTH(K),NXT,NXTARC(J)
       IF(J) 510,550,240
 510   J=-J
       GO TO 201
C
C-----MERGING PROCESS TO CORRESPONDING ARCS
C
 550   DO 651 K=1,MAXK
       LST=0
       NA=TARC(K)
       I=IEND(NA)
C      FIND EAGLE DISTANCE
       TEAGLE(K)=TEAGLE(K)+TVCOST(NA)
       IF(TEAGLE(K).GT.EAGLE(I)) GO TO 650
       EAGLE(I)=TEAGLE(K)
       TB(I)=TTBPTH(K)
       TBARC(I)=NA
C      REFERENCE FIRST BAR PATH FOR BOTH TREES
 650   JT=TFBPTH(K)
       JI=FBPTH(I)
       ISFIR=1
C      TEMPORARY TREE
```

```
 660   IF(JT.EQ.0) GO TO 667
       BARDIS(JT)=BARDIS(JT)+TVCOST(NA)
       IF(BARDIS(JT).GE.EAGLE(I)) GO TO 665
       BART=BARDIS(JT)
       GO TO 670
 665   JT=0
 667   BART=EAGLE(I)
 670   IF(ISFIR.EQ.0) GO TO 700
       ISFIR=0
C      EXISTING TREE
 680   IF(JI.EQ.0) GO TO 687
       IF(BARDIS(JI).GE.EAGLE(I)) GO TO 685
       BARI=BARDIS(JI)
       GO TO 700
 685   JI=0
 687   BARI=EAGLE(I)
       GO TO 700
C      MAKE PERMANENT LABEL BY EITHER TEMPORARY OR
C      EXISTING LABELS
 700   IF(BART.LE.BARI) GO TO 800
C-----PICK EXISTING LABELS
       IF(LST.EQ.0) FBPTH(I)=JI
C      MAKE SURE WHERE LST STAYS
       IF(LST.NE.0) NPOUT(LST)=-JI
 702   LST=JI
       JI=NPOUT(LST)
       IF(JI) 710,710,702
 710   JI=-JI
       GO TO 550
C-----PICK TEMPORARY LABELS
 800   IF(BART.GE.EAGLE(I)) GO TO 840
       IF(LST.EQ.0) FBPTH(I)=JT
       IF(LST.NE.0) NPOUT(LST)=-JT
 802   LST=JT
       JT=NPOUT(LST)
       IF(JT) 810,810,802
 810   JT=-JT
       GO TO 660
 840   IF(LST.NE.0) NPOUT(LST)=0
C      PRINT*,I,TB(I),TBARC(I),EAGLE(I),FBPTH(I)
 850   CONTINUE
C      PRINT*,(J,NPOUT(J),NXTARC(J),BARDIS(J),J=1,NUMPTH)
C      PRINT*,(K,IGROW(K),LSTPTH(K),K=1,MAXK)
C
C-----MOVE TO NEXT NODE
       ND=ND+1
       GO TO 99
C-----BACK TRACING PROCESS
C
 899   NUMNEW=NUMPTH+1
```

```
C       PRINT*,"NUMPTH NUMNEW  EALGE ",NUMPTH,NUMNEW,
C         EAGLE(NUMNODE)
        IF(EAGLE(NUMNODE).EQ.INFIN) GO TO 999
        NUM1(NUMNEW)=NUM1(NUMPTH)+NUMELM(NUMPTH)
        NUMPTH=NUMPTH+1
        IF(NUMPTH.EQ.1)NUM1(NUMPTH)=1
        LLL=NUM1(NUMPTH)
C       PRINT*,"LLL ",LLL
        I=NUMNODE
        LMAX=0
  900   NA=TBARC(I)
        LMAX=LMAX+1
        TIPTH(LMAX)=NA
        NXTI=IBEG(NA)
        IF(NXTI.EQ.1) GO TO 960
        IF(TB(I).NE.0) GO TO 950
C-----FOLLOW EAGLE
        I=NXTI
        GO TO 900
C-----FOLLOW FORBIDDEN PATH
  950   J=TB(I)
        LL=NUM1(J)
  960   JA=IPTH(LL)
        II=IEND(JA)
        IPTH(LLL)=IPTH(LL)
        LLL=LLL+1
        IF(II.EQ.NXTI) GO TO 980
        LL=LL+1
        GO TO 960
  980   L=LMAX
  985   IPTH(LLL)=TIPTH(L)
        L=L-1
        LLL=LLL+1
        IF(L.EQ.0) GO TO 990
        GO TO 985
  990   NUMELM(NUMPTH)=LLL-NUM1(NUMPTH)
        ILAS=NUM1(NUMPTH)+NUMELM(NUMPTH)-1
        IFIS=NUM1(NUMPTH)
        IF(EAGLE1*EAGLE(NUMNODE).LE.TOL(2)) GO TO 995
C
C-----PRINT OUTPUT
C       ARC INPUT DATA
        GO TO 996
995     NUMELM(NUMPTH)=0
        NUMPTH=NUMPTH-1
        GO TO 998
  999   IF(LP.GE.4) WRITE(6,116)
  116   FORMAT(1H ,5X,"NO FEASIBLE SOLUTION EXIST"/1X)
C998    FIN=SECOND(P)
C       TIME=INT((FIN-START)*1000+.5)/1000.
```

```
C       ATIME=ALOG(TIME)
998     CONTINUE
        EAGLE1=EAGLE(NUMNODE)
        RETURN
        END
```



```
C
      SUBROUTINE TABLX
C
C-----SUBROUTINE PROVIDES BASIS TABLEAU
C
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *  OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGJB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,150)
      INTEGER UBND
      DO 100 NA=1,NUMARC
      IW=IARTYP(NA)
      IF(IW.LE.0) GO TO 40
      CALL INVCOL(IW)
      IF(LP.GE.5)
     *PRINT*,"W-INV",("/",COL(J),J=1,NUMARC)
      GO TO 100
40    DO 45 J=1,NUMARC
45    COL(J)=0.
      IF(IW.EQ.-2) COL(NA)=1.
      IF(IW.EQ.-3) COL(NA)=-1./UBND(NA)
      IF(LP.GE.5)
     *PRINT*,"S-YINV",("/",COL(J),J=1,NUMARC)
100   CONTINUE
```

```
      IF(LP.GE.5)
     *PRINT*,"NEW RHS",("/",RHS(I),I=1,NUMARC)
      IF(NUMPTH.LE.0) GO TO 900
      IMAX=NUM1(NUMPTH)+NUMELM(NUMPTH)-1
      IF(LP.GE.4)
     *PRINT *,"IPTH",("/",IPTH(I),I=1,IMAX)
      IF(LP.GE.4)
     *PRINT *,"NUM1",("/",NUM1(I),I=1,NUMPTH)
      IF(LP.GE.4)
     *PRINT *,"NUMELM",("/",NUMELM(I),I=1,NUMPTH)
900   RETURN
      END




C
      SUBROUTINE UPDATE(ICOL,JSMAL)
C
C-----SUBROUTINE HANDLE UPDATING OF ENTERING
C     & LEAVING BASIC
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *   ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *   FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUMELM(100),NUMNODE,
     *   IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *   NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *   IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *   SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *   ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *   OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *   SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,1500)
      INTEGER UBND
      NR=IABS(JSMAL)
      IARBP=IARTYP(NR)
      IF(ISENTER.EQ.1) GO TO 50
      IF(IYENTER.EQ.1) GO TO 60
      NW=IP1BW(ICOL)
      IBEGG=NUM1(NW)
      IENDD=IBEGG+NUMELM(NW)-1
```

```
      IF(LP.GE.4)
     *PRINT *,"W ENTERING",("/",IPTH(I),I=IBEGG,IENDD)
      W(NW)=W(NW)+SMAL
      RHS(NR)=W(NW)
      NUMBTH=NUMBTH+1
C-----GET IARTYP(NR)
      IARTYP(NR)=NUMBTH
      KARTYP(NUMBTH)=NR
      IB1=IP1BW(NUMBTH)
      IP1BW(NUMBTH)=IP1BW(ICOL)
      IP1BW(ICOL)=IB1
      GO TO 70
50    S(ICOL)=S(ICOL)+SMAL
      RHS(NR)=S(ICOL)
      IARTYP(NR)=-2
      IF(ICOL.NE.NR) GO TO 65
      GO TO 70
60    IARTYP(NR)=-3
      Y(ICOL)=1.-SMAL
      IF(IYBSTS(ICOL).EQ.1)Y(ICOL)=SMAL
      RHS(NR)=Y(ICOL)
      IYBSTS(ICOL)=3
      IF(ICOL.EQ.NR) GO TO 70
C-----INTERCHANGE ROWS
65    IW=IARTYP(ICOL)
      KARTYP(IW)=NR
      IARTYP(ICOL)=IARTYP(NR)
      IARTYP(NR)=IW
      RH=RHS(ICOL)
      RHS(ICOL)=RHS(NR)
      RHS(NR)=RH
      JMULT=1
      IF(JSMAL.LE.0)JMULT=-1
      JSMAL=JMULT*ICOL
      DO 20 I=1,NUMBTH
      PINV(I,IW)=SCOL(I)
C     PRINT *,"NEW PINV",("/",PINV(I,J),J=1,NUMBTH)
20    CONTINUE
70    OBJ=OBJ-ZC*SMAL
      IF(IARBP.NE.-3) GO TO 72
C-----SET LEAVING Y STATUSES
      Y(NR)=0.
      IYBSTS(NR)=1
      IF(JSMAL.GT.0) GO TO 72
      Y(NR)=1.
      IYBSTS(NR)=2
      JSMAL=-JSMAL
72    IF(LP.GE.4)
     *PRINT*,"NR IARTYP(NR) RHS(NR) ",NR,IARTYP(NR),RHS(NR)
      IF(IARBP.GT.0)CALL WLEAVE(IARBP)
```

```
75      RETURN
        END
```



```
C
        SUBROUTINE WLEAVE(ID)
C
C-----SUBROUTINE HANDLE DELETING W FROM PINV & COMPLETE
C       DELETION OF IPTH LIST
        COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
       *  ,
       *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
       *  FCOST(500),
       *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
       *  IWENTER,
       *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
       *  NUMNEW,
       *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
       *  IARBP,
       *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
       *  SCOL(500),
       *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
       *  ,
       *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
       *  OBJ,ITLP,
       *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
       *  SMAL,ZO,LP
        COMMON B3(60),BBTAB(4,150)
        INTEGER UBND
C-----DELETE A ROW & A COLUMN FROM PINV
        DO 10 I=1,NUMBTH
        DO 10 J=1,NUMBTH
        IF(I.GT.ID) GO TO 15
        IF(J.GT.ID) PINV(I,J-1)=PINV(I,J)
        GO TO 10
15      IF(J.LT.ID) PINV(I-1,J)=PINV(I,J)
        IF(J.GT.ID) PINV(I-1,J-1)=PINV(I,J)
10      CONTINUE
C-----CHECK W TYPE
        NWR=IP1BW(ID)
        IBEGG=NUM1(NWR)
        IENDD=IBEGG+NUMELM(NWR)-1
        IF(LP.GE.4)
       *PRINT *,"W LEAVING",ID,NWR,("/",IPTH(I),I=IBEGG,IENDD)
        IF(IPSTS(NWR).EQ.3) GO TO 30
C-----DELETE W FROM IPTH
```

```
      CALL DELP(ID)
      GO TO 60
30    IPSTS(NWR)=1
      IF(ID.EQ.NUMBTH)GO TO 50
C-----TEMPORARILY ASSIGN IPTH TO LEAVE
      IP1BW(NUMPTH+1)=IP1BW(ID)
C-----SLIDE TO LEFT FOR (ID+1)IPTH
      IMIM=ID+1
      DO 35 I=IMIM,NUMBTH
      IP1BW(I-1)=IP1BW(I)
      KARTYP(I-1)=KARTYP(I)
      NA=KARTYP(I-1)
      IARTYP(NA)=I-1
35    CONTINUE
C-----ASSIGN IPTH TO LEAVE BACK TO NUMBTH' IPTH
      IP1BW(NUMBTH)=IP1BW(NUMPTH+1)
C-----REDUCE NUMBTH BY ONE
50    NUMBTH=NUMBTH-1
60    CONTINUE
      RETURN
      END




C
      SUBROUTINE XSOL
C-----SUBROUTINE HANDLES TO OBTAIN NODE-ARC SOLUTION
C
      COMMON/AAPDAT/X(500),XINCUM(500),XLAGR(500),YLAGR(500)
     *  ,
     *IBEG(500),IEND(500),IPTH(5000),IYENTER,ISENTER,
     *  FCOST(500),
     *VCOST(500),UBND(500),LBND(500),NUME_M(100),NUMNODE,
     *  IWENTER,
     *NUMPTH,NUMOAR(150),NUM1(100),IFIRARC(150),VLAST,
     *  NUMNEW,
     *TIME,ATIME,TVCOST(500),KARTYP(100),TCOL(500),NUMARC,
     *  IARBP,
     *IARTYP(500),COL(500),RHS(500),IP1BW(100),S(500),INFIN,
     *  SCOL(500),
     *IPSTS(100 ),W(100),T(100),DTIL(100),Y(500),IYBSTS(500)
     *  ,
     *FTIL(500),IYGUB(100),IYELM(100),BETA(100),EPS(100),
     *  OBJ,ITLP,
     *Q1(75),PINV(75,75),ALPA(5000),NUMGUB,NUMBTH,TOL(5),
     *  SMAL,ZC,LP
      COMMON BB(60),BBTAB(4,1500)
```

```
      INTEGER UBND
      DO 10 I=1,NUMARC
      YLAGR(I)=Y(I)
10    XLAGR(I)=0.
      DO 15 IW=1,NUMBTH
      NW=IP1BW(IW)
      NUM=NUM1(NW)
      NUMEL=NUM+NUMELM(NW)-1
      DO 16 I=NUM,NUMEL
      NA=IPTH(I)
C     PRINT*,"NW W(NW) NA  ",NW,W(NW),NA
16    XLAGR(NA)=XLAGR(NA)+W(NW)
15    CONTINUE
      IF(LP.GE.3)
     *PRINT *,"XLAGR ",("/",XLAGR(NA),NA=1,NUMARC)
      RETURN
      END
```

BIOBLIOGRAPHY

1. Balas, E., "An Infeasibility--Pricing Decomposition Method for Linear Programs," Operations Research, 14, 847-873, 1966.

2. Bazaraa, M. S. and J. J. Goode, "The Traveling Salesman Problem: A Duality Approach," School of Industrial and Systems Engineering, Georgia Institute of Technology, 1974.

3. Beale, E. M. L. and R. E. Small, "Mixed Integer Programming by a Branch-and Bound Technique," in: W. A. Kalerick, ed., Proceedings of the IFIP Congress (Spartan Press, East Lansing, MI, 1965).

4. Benders, J. F., "Partitioning Procedures for Solving Mixed Variables Programming Problems," Numerische Methamatik, 4, 238-252, 1962.

5. Bulfin, R. L., "An Algorithm for the Plant Location Problem," Masters thesis in the School of Industrial and Systems Engineering, Georgia Institute of Technology, 1972.

6. Bulfin, R. L. and V. E. Unger, "Computational Experience with an Algorithm for the Locked Box Problem," Proceedings of 1973 Conference of the ACM, 16-19, 1973.

7. Dantzig, G. B. and P. Wolfe, "The Decomposition Algorithm for Linear Programming," Econometrica, 29, 767-778, 1961.

8. Davis, P. S. and T. L. Ray, "Branch and Bound Algorithm for the Capacitated Facilities Location Problem," Naval Research Logistics Quarterly, 16, 331-344, 1969.

9. Deo, N., Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, Inc., 1973.

10. Dijkstra, E. W., "A Note on Two Problems in Connection with Graphs," Numerical Mathematics, 1, 269-271, 1959.

11. Driebeek, N. J., "An Algorithm for the Solution of Mixed Integer Programming Problems,""Management Science, 12, 576-587, 1966.

12. Efroymson, M. A. and T. L. Ray, "A Branch-and-Bound Algorithm for Plant Location," Operations Research, 14, 361-368, 1966.

13. Ellwein, L. B., "Fixed Charge Location-Allocation Problems with Capacity and Configuration Constraints," Technical Report No. 70-2, Department of Industrial Engineering, Stanford University, 1970.

14. Erlenkotter, D., "Facility Location with Price-Sensitive Demands: Private, Public, and Quasi-Public," Working Paper No. 225, Western Management Science Institute, University of California, Los Angeles, 1976.

15. Erlenkotter, D., "A Dual-based Procedure for Uncapacitated Facility Location," Discussion Paper No. 64, Management Science Center, University of California, Los Angeles, 1976, Forthcoming in Operations Research.

16. Ford, L. R. and D. R. Fulkerson, "A Suggested Computation for Maximal Multicommodity Network Flows," Management Science, 5, 97-101, 1958.

17. Frank, R. S., "On the Fixed Charge Hitchcock Transportation Problem," Ph.D. dissertation, The Johns Hopkins University, 1972.

18. Geoffrion, A. M., "Lagrangian Relaxation for Integer Programming," Mathematical Programming Study 2, 82-114, 1974.

19. Geoffrion, A. M. and G. W. Graves, "Multicommodity Distribution System Design by Benders Decomposition," Management Science, 20, 822-844, 1974.

20. Geoffrion, A. M. and R. D. McBride, "The Factorization Approach to Large Scale Linear Programming," Mathematical Programming, 10, 91-110, 1976.

21. Geoffrion, A.M. and R. D. McBride, "Lagrangian Relaxation Applied to Facility Location Problems," Forthcoming in AIIE Transactions.

22. Glover, F., "Compact LP Bases for a Class of IP Problems," Management Science Report Series Report No. 75-18, Graduate School of Business Administration, University of Colorado, 1975.

23. Gray, P., "Mixed Integer Programming Algorithms for Site Selection and Other Fixed-Charge Problems Having Capacity Constraints," Technical Report No. 101, Department of Operations Research and Statistics, Stanford University, 1967.

24. Hansen, P., "Two Algorithms for the Simple Plant Location Problem Using Additive Penalties," Presented at the European Congress of the Econometric Society, Budapest, 1972.

25. Hearn, D. and T. J. Lowe, "A Subgradient Procedure for the Solution of Minimax Location Problems," Research Report No. 76-6, School of Industrial and Systems Engineering, University of Florida, 1976.

26. Held, M. and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees," Operations Research, 18, 1138-1162, 1970.

27. Held, M. and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II", Mathematical Programming, 1, 6-25, 1971.

28. Held, M., P. Wolfe, and H. P. Crowder, "Validation of Subgradient Optimization," Mathematical Programming, 6, 62-88, 1974.

29. Hillier, F. S., "An Optimal Bound and Scan Algorithm for Integer Linear Programming," Technical Report No. 3, Department of Industrial Engineering, Stanford University, 1966.

30. Jarvis, J. J., "On the Equivalence Between the Node-Arc and Arc-Path Formulation for the Multicommodity Maximal Flow Problem, Naval Research Logistics Quarterly, 16, 525-529, 1969.

31. Jarvis, J. J., V. E. Unger, R. L. Rardin, R. W. Moore and C. C. Schimpeler, "Optimal Design of a Regional Water Quality Waste Management System: A Fixed Charge Network Flow Model," 1975.

32. Kennington, J. L., "Fixed-Charge Transportation Problem: A Group Theoretical Approach," Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1973.

33. Khumawala, B. M., "An Efficient Branch and Bound Algorithm for the Warehouse Location Problem," Management Science, 18, 718-731, 1972.

34. Land, A. H. and A. G. Doig, "An Automatic Method for Solving Discrete Programming Problems," Econometrica 28, 497-520, 1960.

35. Murty, K. G., "Solving the Fixed Charge Problem by Ranking Extreme Points," Operations Research, 16, 268-279, 1968.

36. Neebe, A. W. and M. R. Rao, "A Subgradient Approach to the m-Median Problem," Technical Report No. 75-12, Presented at the Joint National Meeting of ORSA/TIMS in Las Vegas, 1975.

37. Poljak, B. T., "A General Method of Solving Extremum Problems," Soviet Mathematics Doklady, 8, 593-597, 1967.

38. Ramsay, T. E., "Computational Comparison of Two Integer Programming Formulations," School of Industrial and Systems Engineering, Georgia Institute of Technology, December 1976.

39. Rardin, R. L., "Group Theoretical and Related Approaches to Fixed Charge Problems," Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, December 1973.

40. Rardin, R. L. and V. E. Unger, "Some Computationally Relevant Group Theoretic Structures of Fixed Charge Problems," Mathematical Programming, 10, 379-400, 1976.

41. Schrage, L., "Implicit Representation of Variable Upper Bounds in Linear Programming," Working Paper No. 217, Western Management Science Institute, University of California, Los Angeles, 1974.

42. Seshu, S. and M. B. Reed, Linear Graphs and Electrical Networks, Addison-Wesley Publishing Company, Inc., 1961.

43. Spielberg, K., "On the Fixed-Charge Transportation Problem," Proceedings of the 19th National Conference of the ACM, A1. 1-1 to A1. 1-13, 1964.

44. Tomlin, J. A., "Minimum-Cost Multicommodity Network Flows," Operations Research, 14, 45-51, 1966.

45. Tomlin, J. A., "An Improved Branch and Bound Method for Integer Programming," Operations Research, 19, 1070-1075, 1971.

46. Tompkins, C. J., "Group Theoretic Structure in the Fixed Charge Transportation Problem," Ph.D. Dissertation, School of Indùstrail and Systems Engineering, Georgia Institute of Technology, 1971.

47. Williams, H. P., "Experiments in the Formulation of Integer Programming Problems," Mathematical Programming Study 2, 180-197, 1974.

48. Wollmer, R. D., "Multicommodity Network with Resource Constraints: The Generalized Multicommodity Flow Problem," Networks, 1, 245-263, 1972.

VITA

The author was born in Japan on July 18, 1940. He graduated from the Kyung-Nam High School in 1958 and immediately entered the Korean Naval Academy. There Mr. Choe was awarded a Bachelor of Science degree (1962) and was commissioned as ensign to begin his professional career as a naval officer in the Korean Navy. In 1964 he left Korean Fleet to have gunnery officer training at the U. S. Naval Training Center, Bainbridge, Maryland. He served as gunnery officer on a destroyer and rocket-equipped landing ship for more than two years.

He returned to the United States for his graduate study in the U. S. Naval Postgraduate School in Monterey, California and obtained his Masters degree in Operations Research in September 1970.

After working several years as a senior member of the Systems Analysis Group for the Korean Navy, he came back to the United States to enter the School of Industrial and Systems Engineering at the Georgia Institute of Technology.

For his services in the Korean Navy he received citations from the Minister of National Defense (1963), the Mayor of Seoul Metropolitan City (1971), and Chief of Naval Operations (1973).

While at Georgia Tech, he pursued a Doctoral program in Operations Research while working as a Graduate Research Assistant. He was awarded the Ph.D. degree in 1977. The author is currently working as a senior member of the Systems Analysis Group for the Korean Navy.